PATENT ABSTRACTS OF JAPAN

(11)Publication number : 06-083682

(43)Date of publication of application : 25.03.1994

(51)Int.Cl. G06F 12/00
G06F 12/00

(21)Application number : 03-242865

(22)Date of filing : 14.06.1991

(71)Applicant : DIGITAL EQUIP CORP <DEC>

(72)Inventor : LOMET DAVID B
SPIRO PETER M
JOSHI ASHOK M
RAGHAVAN ANANTH
RENGARAJAN TIRUMANJANAM K

(30)Priority

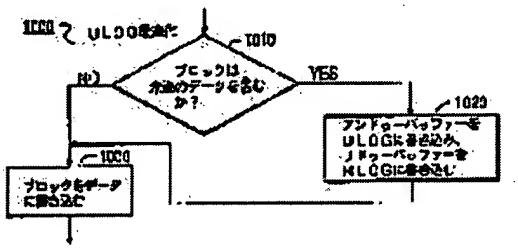
Priority number	Priority date	Priority country
90 548720	29.06.1990	US
90 546306	02.07.1990	US

(54) METHOD AND DEVICE FOR MAXIMUM USE OF UNDO LOG

(57)Abstract:

PURPOSE: To minimize information to be recorded in an undo transaction at the time of the occurrence of a crash or a failure.

CONSTITUTION: It is discriminated whether a write block includes data which is not committed or not (1010) for the purpose of optimizing an undo log with a write-ahead log protocol. If it includes this data, an undo buffer is written in the undo log and a redo buffer is written in a redo log (1020). Thereafter or if it doesn't include this data (1010), the block is written in a permanent storage device (1030).



LEGAL STATUS

[Date of request for examination]	14.06.1991
[Date of sending the examiner's decision of rejection]	
[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]	
[Date of final disposal for application]	
[Patent number]	2501152
[Date of registration]	13.03.1996
[Number of appeal against examiner's decision of rejection]	
[Date of requesting appeal against examiner's decision of rejection]	
[Date of extinction of right]	

* NOTICES *

1.This document has been translated by computer. So the translation may not reflect the original precisely.
2.**** shows the word which can not be translated.
3.In the drawings, any words are not translated.

CLAIMS

[Claim(s)]
[Claim 1] It is the data processing system which has two or more nodes and the non-volatile storage medium divided into the section. Two or more nodes change with transactions to a section, and each transaction consists of a series of change made by at least one node into at least one section. In the data processing system it will be commissioned each transaction if record of change performed by directions of completion of a transaction and a transaction is certainly memorized by the storage medium, otherwise it is not commissioned data processing system The memory in which one of the beginnings of two or more nodes holds the copy of at least one section, A processing means to connect with memory and to change to the copy of at least one section in memory, It has at least one undo buffer including the sequential list of change made by the copy of at least one section in memory with a processing means. Each undo buffer corresponds to change made by the transaction which changed with first nodes, and it is not commissioned the transaction. A redoing buffer including the sequential list of change made by the copy of at least one section in memory with the processing means in a corresponding node, A storage means to connect with memory and to store the copy of at least one section in a storage medium, It connects with an undo buffer and a redoing buffer. Some of undo buffers and redoing buffers are selectively stored in a storage medium. Data processing system characterized by having the log management tool which secures carrying out the reconstitution of the effectiveness of all change of the transaction removed the effectiveness of all change of the transaction it is not commissioned a transaction, and it was commissioned the transaction.
[Claim 2] A log management tool is a node characterized by having a means to store the content of the restoration buffer in a storage medium before storing change from the transaction to which a storage means corresponds in a node according to claim 1, and it has not commissioned a transaction.
[Claim 3] It is the node characterized by having a means to certainly store in a storage medium all the change to the first transaction by which the log management tool was recorded on the repetitive buffer in the first node according to claim 1, and directions of completion of the first transaction, and to make it commission the first transaction by it.

[Translation done.]

* NOTICES *

1.This document has been translated by computer. So the translation may not reflect the original precisely.
2.**** shows the word which can not be translated.
3.In the drawings, any words are not translated.

DETAILED DESCRIPTION

[Detailed Description of the Invention]
[Background of the Invention] Generally this invention relates to the activity of the field of recovery from the crash in a common disc system, and the log especially in this recovery. Data may be lost when all computers crash. There are some a certain systems which have especially high possibility of losing data by failure or crash. This is for those systems to transmit the data of a large quantity between a disk and processor memory. The reason common to losing data is imperfect transmission at the time of sending data to an endurance mold storage system (for example, disk) from a non-bearing mold storage system (for example, processor memory). If processing is performed at the time of crash, imperfect transmission will often arise. Generally processing includes transmission of a series of storage between two storage systems (or modification). In case the address of loss and its recovery of data is carried out, an important concept "performs" processing. [intend] Processing is performed when it is guaranteed to some extent that all the effectiveness is stable in endurance mold storage. a thing required for recovery when crash produces a process required for recovery after processing when crash arises before processing was performed -- things -- ** Recovery is a process which corrects a database so that a perfect system can be resumed at the event of the known request. The class of recovery needed is based on the reason for loss of data, though natural. If the system of a computer crashes, the recovery means needs to be recoverable to the condition which is in agreement with what produced endurance mold storage, such as a crash system, for example, a disk etc., by processing performed last time. If endurance mold storage crashes (it is called medium failure), it is necessary to reproduce on a disk the data stored in the recovery. Many means for recovering database system are using the log. It is only the list of action of the order of time of day in which it is shown what kind of change, as for the log, was made at least at the database in the case of the database and in what kind of sequence those change was made again. Therefore, a log is changed into the condition of the request which had the database known by the computer system, and a database is applicable to modification of redoing (REDO) or undo (UNDO) after that. Redoing means rerun and undo means an invalid. However, in the system arrangement many computer systems of whose called a node access the set of a common disk, it is hard to manage a log. The type of this configuration is called a "cluster" or a common disc system. The system which makes data access a node in such a system is called a data shared bus system. A data shared bus system transmits data and, thereby, is sent to the computer which the data block itself requires from a disk. The functional transmitting system known more widely as a "partition" system reversely transmits the set of operation to the computer specified as a "server" of a partition of data. Then, a server returns the side which performs operation and requires the result. In a partition system, each part of data can belong in the local memory of one or less node like a single node or concentrated system. Moreover, both a partition system and concentrated system need only action memorized by the single log. It is important that recovery of data can be performed only based on the content of one log here. On the other hand, the transmitting system of the distributed data is not centralized, but the same data belong in the local memory of many nodes, and can update it from those nodes. Thereby, much node logging actions arise for the same data. In order to avoid the problem of action of having many logs for the same data, a data

transmitting system requires that record of the log for data should be returned to a single log with the charge which records the recovery information of data. However, the independently new source of a system is required of such remote logging. Because, it is for writing another message which has record of a required log besides I/O in a log. Furthermore, delay can become large by waiting for a receipt from a logging computer. It is also decreasing the capacity the response time's not only increasing by this, but a user's being able to access simultaneously to the same database and being to it. It is using simultaneously the common log written in by turns as an option. However, since this also needs another message for adjustment, costs start too much. These troubles are important for the address. because, it takes -- it carries out for taking and is because the data shared bus system is more desirable than a partition system. For example, since a data shared bus system can make the period data extended to the workstation store and partial processing of data can be performed with the high level by this, the data shared bus system is important for a workstation and engineering design application. Furthermore, since many nodes access data simultaneously, and a data shared bus system manages some partial data itself and can share other data with other host computers and a workstation, it has a fault tolerance and load balance nature essentially. Therefore, the object of this invention is making management of a redoing log easy by removing undo information from a redoing record. Moreover, other objects of this invention are making management of undo information easier by abandoning undo information at the time of transaction activation. Furthermore, other objects of this invention are pressing down the information which must be made to record on an undo transaction to the minimum, when crash or failure occurs. Epitome this invention of II. invention avoids the trouble of the advanced technology by maintaining sufficient information from redoing and an undo buffer so that all modification of the transaction which is not performed can be removed, and modification from the performed transaction can be reproduced and the amount of records of the undo buffer to an undo log can be stopped to the minimum. Furthermore, since action is not performed, effectiveness is maintainable by holding the count of action in a transaction. In the data processing system which has in a detail two or more nodes and the endurance mold storage divided for every section Said two or more nodes change to said section by the transaction. Said each transaction by at least one node It has a series of modification made to at least one section. Said each transaction It performs, when the display of completion of the record of modification made by this transaction and this transaction is certainly memorized by the storage. It does not perform, when not memorizing certainly. The first one of said two or more of the nodes It connects with the memory and; this memory holding the copy of at least one section. As opposed to at least one copy in said memory A processing means to change to the copy of at least one section in the memory; with said processing means At least one undo buffer and each; this undo buffer which have a sequential list of made modification correspond to modification made to a different transaction by which the first node does not perform for this node and each undo buffer. The record buffer which has a sequential list of modification made to the copy of at least one section in memory by the processing means in a corresponding node; It connects with said memory. The storage means for returning and memorizing the copy of at least one section to said storage; The effectiveness of all modification of the transaction which is not performed is removable, And in order to ensure that all modification of the performed transaction is reproducible It is characterized by consisting of a log management tool connected with the undo buffer and redoing buffer which memorize selectively the parts of said undo buffer and a redoing buffer to said storage, and;. The accompanying drawing which is a part of this application shows the suitable example of this invention, and explains the principle of this invention with this description.
[Best Mode of Carrying Out the Invention] The suitable example of this invention is explained in full detail below. The example is illustrated to an accompanying drawing.
A. The structure-of-a-system element system 100 is an example of the storage system which can be used for carrying out this invention. A system 100 has nodes 110, 120, and 130, and these all access the common disc system 140. Nodes 110, 120, and 130 have processors 113, 123, and 133, respectively, in order to perform the below-mentioned storage and recovery roux teens. Nodes 110, 120, and 130 have memory 118, 128, and 138 further, respectively, in order to offer at least two functions. One side operates as partial memory for a corresponding processor among these functions, and another side is

exchanged for said disc system 140, and holds data. The part of the memory used for the data exchange is called a cache. Generally a cache is a non-bearing mold storage system. The common disc system 140 will also point out the endurance mold storage system considered that the content of storage is permanence, even if all the permanent storage all [parts or] crash. Although this endurance mold system has adopted the magnetic disc system traditionally, a permanent storage system can also use an optical disk or a magnetic tape system. Moreover, the permanent storage system used for carrying out this invention is not limited only to the architecture (design specification of a computer) shown in drawing 1 . For example, this permanent storage system can also use the disk of the shoes combined with a different node, respectively, and these nodes may be connected with the network of some types. the grasshopper to which other parts of a permanent storage system are called an archive (archive) storage system -- it is the rise system 150. When the data in a permanent storage system read vocabulary called an archive storage system and it becomes impossible, generally it is used to mention the storage system used for the information which restores the content of the permanent storage system. For example, when the common disc system 140 causes a failure to a medium, in order to restore the system 140, the tape system 150 can be used. Although the archive storage system is usually using the magnetic tape system, the MAG or an optical disc system can also be used. The data of a system 100 are usually memorized by block and it is the object which can recover this system. Generally, a block can operate, only when it is in the cache of a node with them. Drawing 2 shows an example of the blocks 210, 220, and 230 of some disks 200. Generally one block has the pagination of the integer of a permanent storage system. For example, in drawing 2 , the block 210 has pages 212, 214, 216, and 218.

B. Most database system is using the log aiming at recovery as log above-mentioned. A log is usually memorized in a permanent storage system. When a node updates a permanent storage system, a node memorizes the log record which describes updating in the buffer in the cache of the node. Although the log of three types is assumed in a permanent storage system in the suitable example of this invention, there is only a buffer of two types in the cache of each node. Said logs are a redoing log (RLOG), an undo log (RLOG), and an archive log (ALOG). Moreover, said buffers are a redoing buffer and an undo buffer. The example of ULOG is shown in drawing 3 and drawing 4 and the example of ALOG are shown for the example of RLOG in drawing 5 . The configuration of a redoing buffer is similar to RLOG, and the configuration of an undo buffer is similar to ULOG. The sequence number (LSN) of a log is the address or the relative location of a record in a log. Each log shows LSN to the record in the log.

1. As shown in RLOG (redoing log) drawing 3 , it is the suitable example of the sequential file used in order that RLOG300 may record the information about modification, and permit that the information repeats the specific operation performed while those modification is shown. If restored to the condition that action on which the block was recorded once was performed, it is necessary to make such operation repeat during a recovery plan generally. As shown in drawing 3 , RLOG300 has records 310, 302, and 310, and each record has some attributes. The TYPE attribute 320 identifies the type of a corresponding RLOG record. As an example of the type with which RLOG records differ, they are a redoing record, an amendment log record, and a record about activation. These records are explained below. The TID attribute 325 is the only identifier for the transaction relevant to a current record (current record). This attribute is used for the assistance for looking for the record of ULOG corresponding to a current RLOG record. The BSI attribute 300 is an identifier before a condition (before state identifier). This identifier is explained in full detail below. If it says simply, this BSI shows the value of the identifier of the condition for the version of the block before a correction change is made by the corresponding transaction. The BID attribute 335 identifies the block in which a correction change was made by updating corresponding to a RLOG record. REDO The DATAM attribute 340 describes the property of action to correspond, and offers sufficient information for repeating the action. the vocabulary "updating" -- the inside of this description -- a wide sense -- and it is used for the same semantics as the vocabulary "action." If it is strictly called action, not only modification of a record but assignment and disconnection of insertion of a record, deletion, and a block are included. The LSN attribute 345 identifies the current record of RLOG300 uniquely. It is used for the LSN buffer 345 controlling a

REDOSU can and a RLOG checkpoint by this example so that it may mention later. LSN345 is memorized by neither a RLOG record nor block by this example. That is because it is originally the thing of a proper from the location of the record in RLOG. The object of this invention is that other nodes make each node manage the recovery as independently as possible. For this reason, separate RLOG is related to each node. The relation of the node and RLOG in a suitable example uses the approach of using RLOG which is different in each node. As an option, a node can be made to be able to share RLOG or much RLOG(s) can also be given to each node. however -- the case where RLOG is only for [of one] nodes -- the -- ** -- it is unnecessary in the synchronization containing the message for using RLOG for other RLOG(s) and nodes equally.

2. In ULOG (undo log) drawing 4 , ULOG400 is the suitable example of the sequential file used for recording the information which cancels the operation of a block correctly. It is used for ULOG400 restoring a block to a condition when a transaction starts. Unlike RLOG, ULOG and an action buffer relate to a different transaction, respectively. Therefore, ULOG(s) and those corresponding buffers will disappear, if a transaction is performed, and if a new transaction begins, new ULOG will appear. Other possibility exists. ULOG400 has records 410, 402, and 410, and each record has the two fields.

DESCRIPTION OF DRAWINGS

[Brief Description of the Drawings]
[Drawing 1] The schematic diagram showing the computer system for an example [this invention],
[Drawing 2] Some schematic diagrams of the disk in which a block and a page are shown,
[Drawing 3] The schematic diagram of REDOROGU,
[Drawing 4] The schematic diagram of an undo log,
[Drawing 5] The schematic diagram of an archive log,
[Drawing 6] The flow chart for REDOOPERESHON activation,
[Drawing 7] The flow chart for performing recovery from crash,
[Drawing 8] The flow chart for combining an archive log,
[Drawing 9] The schematic diagram of a dirty block table,
[Drawing 10] The flow chart for performing a light-ahead protocol that the maximum utilization of the activity of an undo log should be carried out,
[Drawing 11] The schematic diagram of an amendment log record,
[Drawing 12] The table of an active transaction,
[Drawing 13] The flow chart for transaction initiation operation,
[Drawing 14] The flow chart of the renewal operation of a block,
[Drawing 15] The flow chart of block write-in operation,
[Drawing 16] The flow chart for the abandonment operation of a transaction,
[Drawing 17] The flow chart for the preparation operation of a transaction,
[Drawing 18] The flow chart for the modification operation of a transaction.

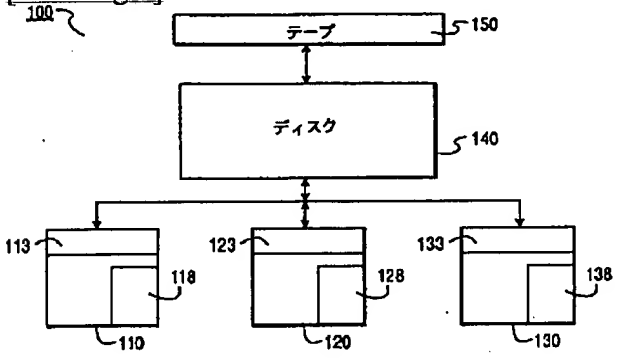
[Translation done.]

* NOTICES *

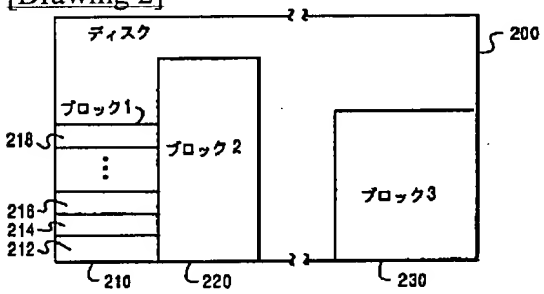
1.This document has been translated by computer. So the translation may not reflect the original precisely.
2.**** shows the word which can not be translated.
3.In the drawings, any words are not translated.

DRAWINGS

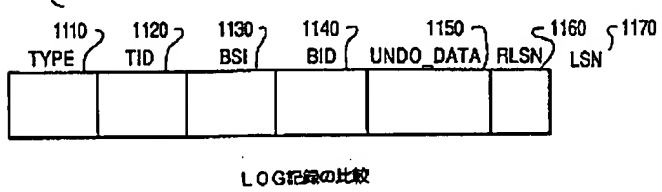
[Drawing 1]



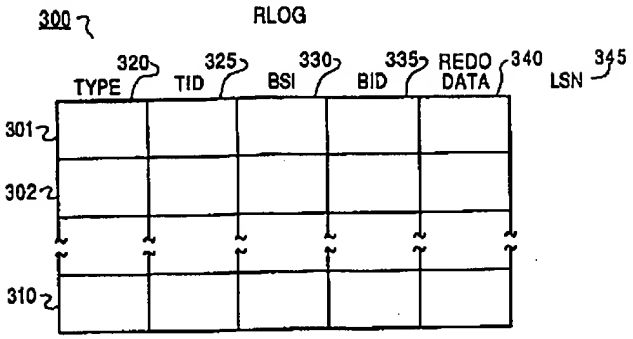
[Drawing 2]



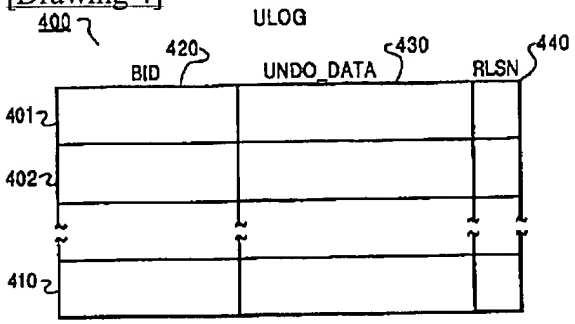
[Drawing 11]



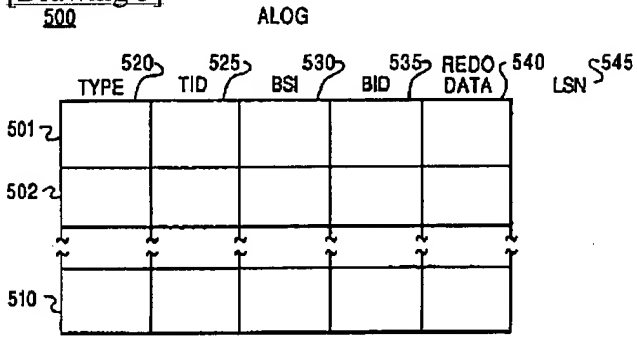
[Drawing 3]



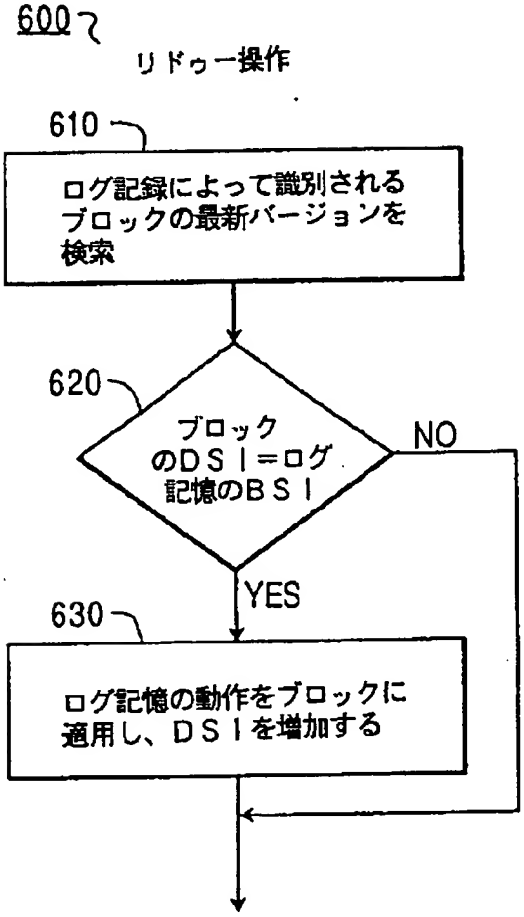
[Drawing 4]



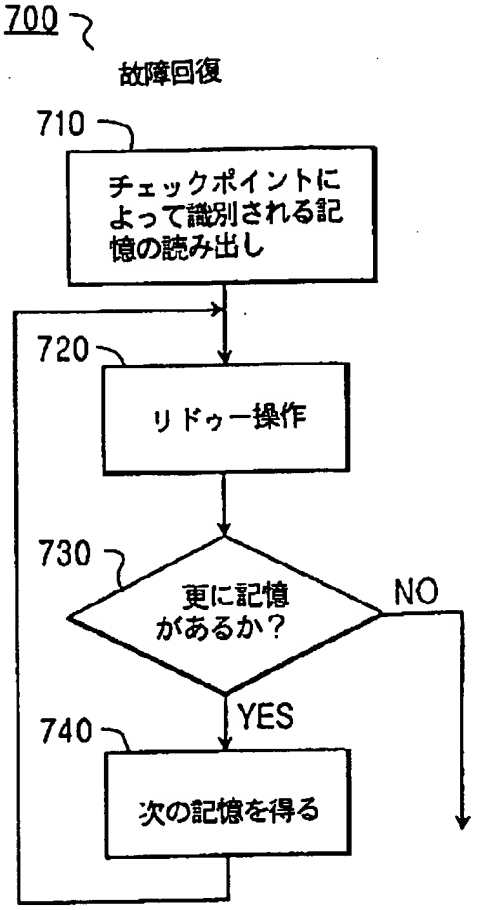
[Drawing 5]



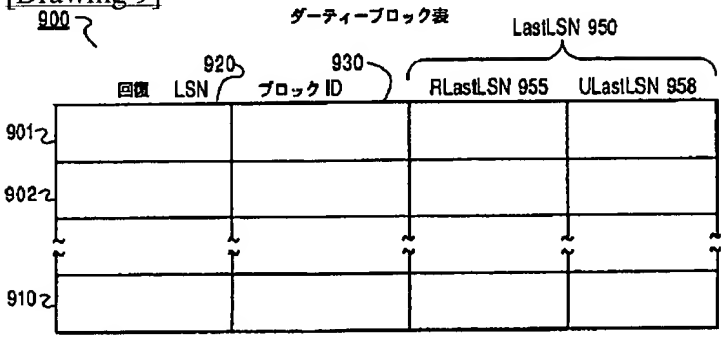
[Drawing 6]


600
リドゥー操作
610
ログ記録によって識別される
ブロックの最新バージョンを
検索
620
ブロック
のDSI＝ログ
記憶のBSI
NO
YES
630
ログ記憶の動作をブロックに
適用し、DSIを増加する

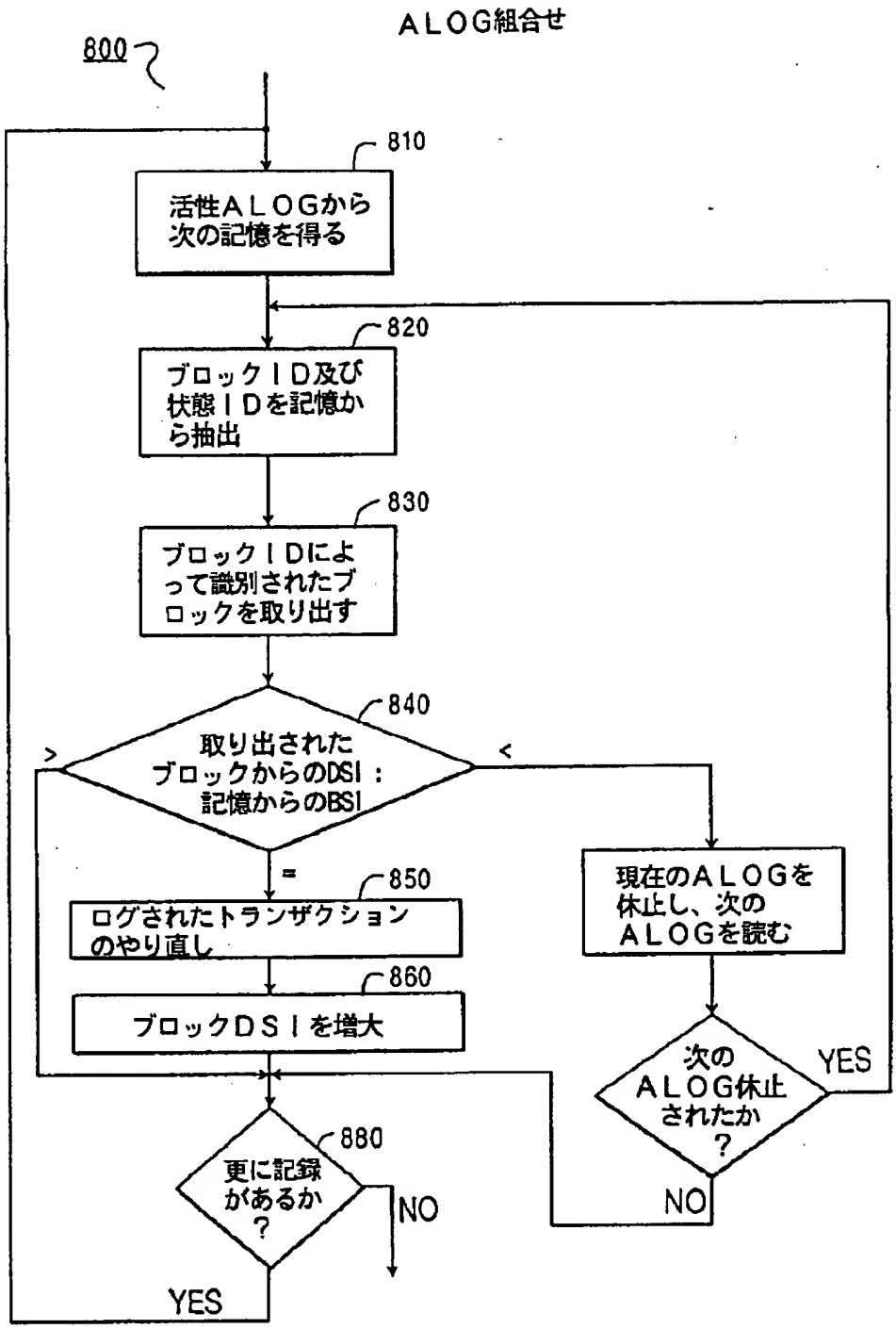
[Drawing 7]



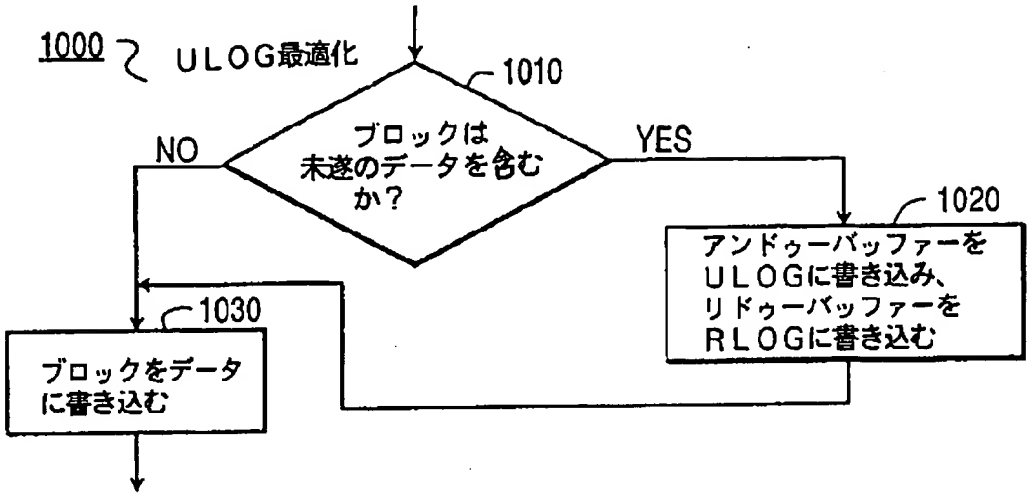
[Drawing 9]



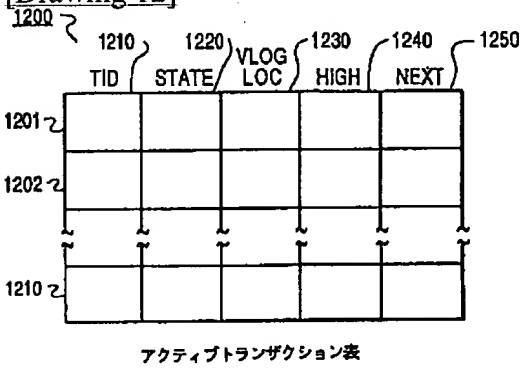
[Drawing 8]


ALOG組合せ
800
810
活性ALOGから次の記憶を得る
820
ブロックID及び状態IDを記憶から抽出
830
ブロックIDによって識別されたブロックを取り出す
840
取り出されたブロックからのDSI：記憶からのBSI
>
<
=
850
ログされたトランザクションのやり直し
860
ブロックDSIを増大
現在のALOGを休止し、次のALOGを読む
次のALOG休止されたか？
YES
NO
880
更に記録があるか？
NO
YES

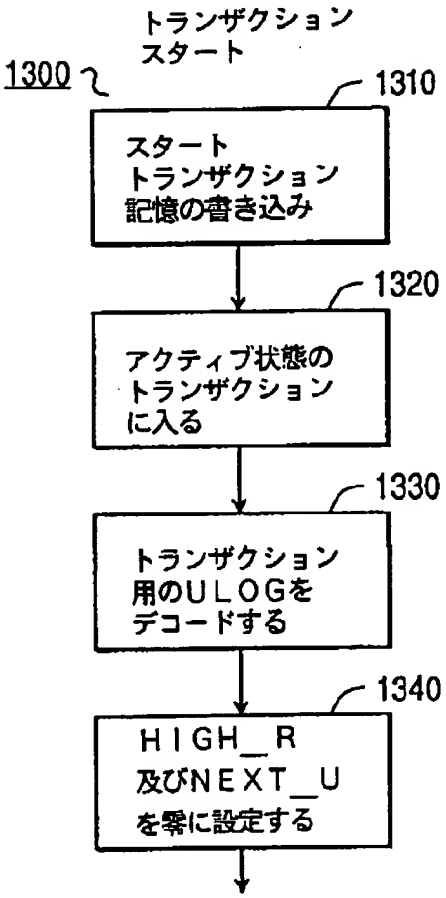
[Drawing 10]



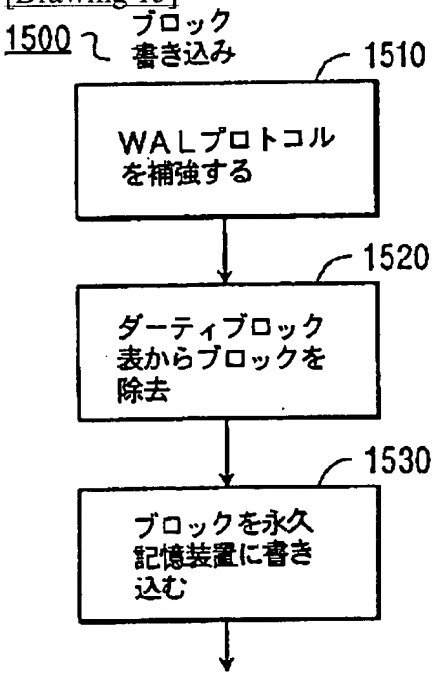
[Drawing 12]



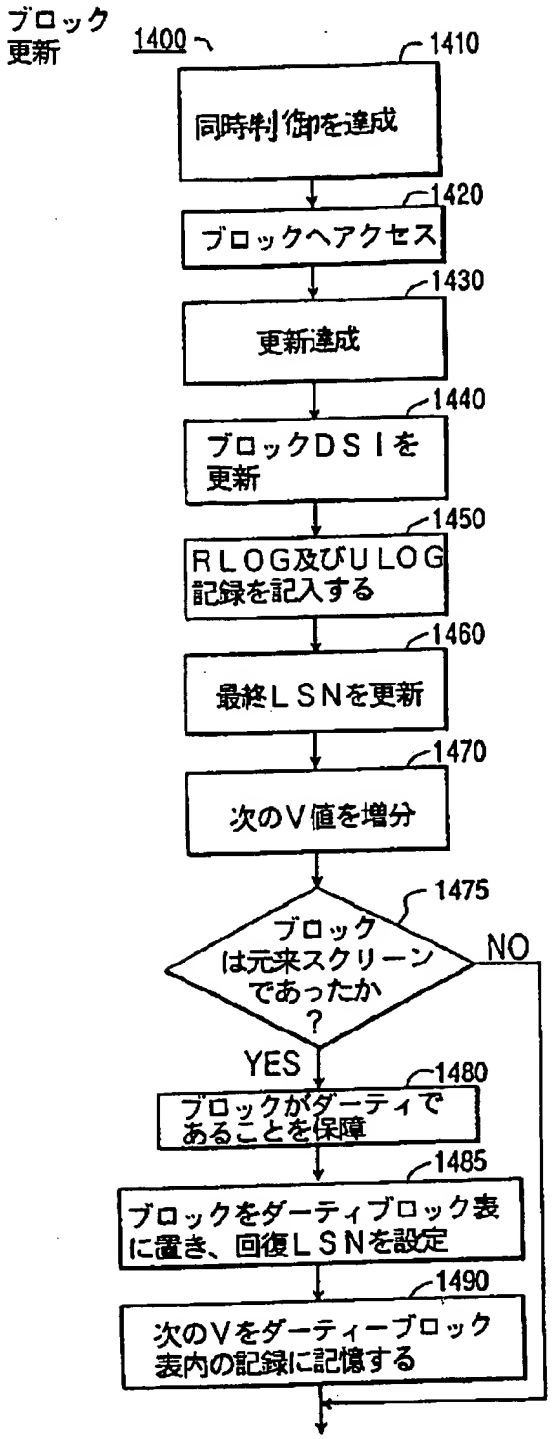
[Drawing 13]



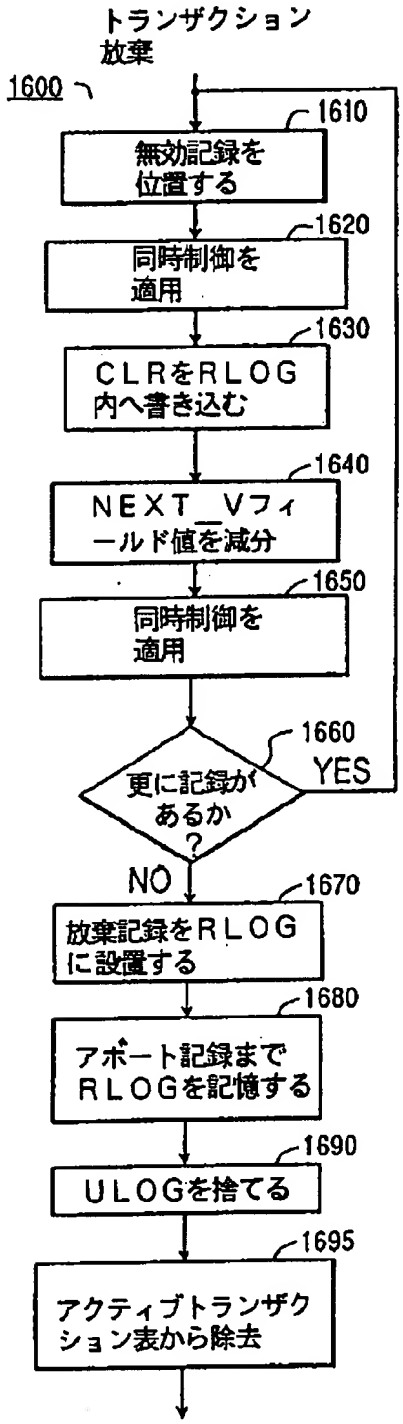
[Drawing 15]



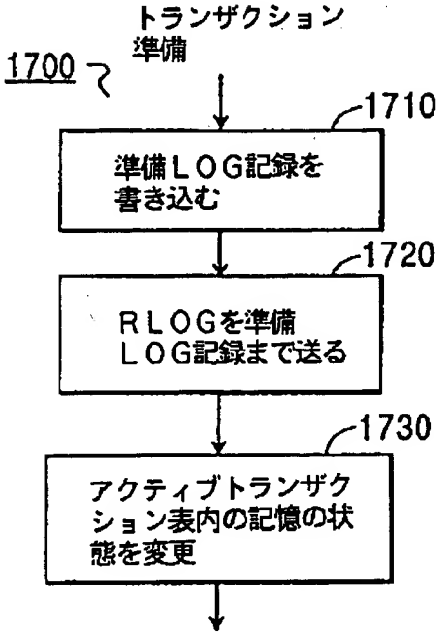
[Drawing 14]


ブロック更新
1400
1410
同時制御を達成
1420
ブロックへアクセス
1430
更新達成
1440
ブロックDSIを更新
1450
RLOG及びULOG記録を記入する
1460
最終LSNを更新
1470
次のV値を増分
1475
ブロックは元来スクリーンであったか？
NO
YES
1480
ブロックがダーティであることを保障
1485
ブロックをダーティブロック表に置き、回復LSNを設定
1490
次のVをダーティーブロック表内の記録に記憶する

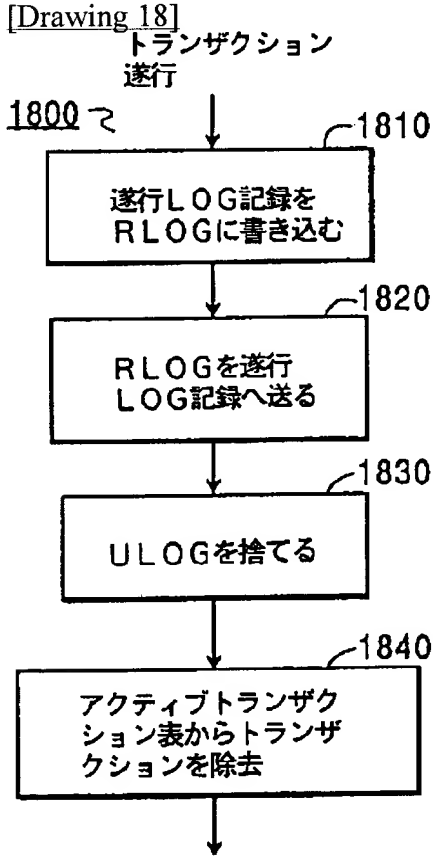
[Drawing 16]


トランザクション
放棄
1600
1610
無効記録を
位置する
1620
同時制御を
適用
1630
CLRをRLOG
内へ書き込む
1640
NEXT_Vフィ
ールド値を減分
1650
同時制御を
適用
1660
更に記録が
あるか
?
YES
NO
1670
放棄記録をRLOG
に設置する
1680
アボート記録まで
RLOGを記憶する
1690
ULOGを捨てる
1695
アクティブトランザク
ション表から除去

[Drawing 17]



[Drawing 18]



[Translation done.]

(19)日本国特許庁（ＪＰ）　(12) 公開特許公報（A）　(11)特許出願公開番号

特開平6－83682

(43)公開日　平成６年(1994)３月25日

(51)Int.Cl.[5]	識別記号	庁内整理番号	ＦＩ	技術表示箇所
Ｇ０６Ｆ　12/00	５１８　Ａ	8526－5B		
	５３１　Ｒ	8526－5B		

審査請求　有　　請求項の数３(全 22 頁)

(21)出願番号　特願平3－242865

(22)出願日　平成３年(1991)６月14日

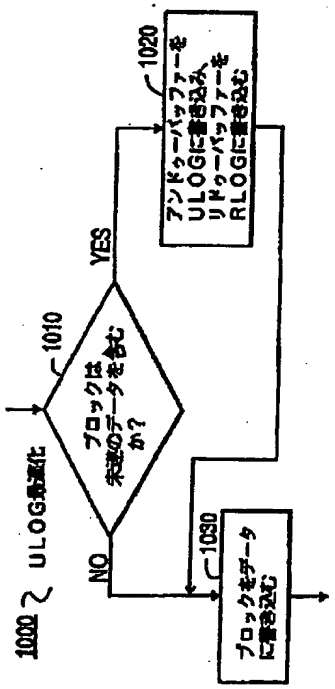
(31)優先権主張番号　５４８７２０
(32)優先日　1990年６月29日
(33)優先権主張国　米国（ＵＳ）
(31)優先権主張番号　５４６３０６
(32)優先日　1990年７月２日
(33)優先権主張国　米国（ＵＳ）

(71)出願人　590002873
ディジタル　イクイプメント　コーポレイション
アメリカ合衆国　マサチューセッツ州　01754メイナード　メイン　ストリート　146
(72)発明者　ディヴィッド　ピー　ロメット
アメリカ合衆国　マサチューセッツ州　01886ウエストフォード　チェリー　レーン　9
(74)代理人　弁理士　中村　稔　(外７名)

最終頁に続く

(54)【発明の名称】　アンドゥーログ使用の最大利用のための方法及び装置

(57)【要約】　(修正有)
【目的】　クラッシュ或は故障が発生した際にアンドゥートランザクションに記録させなくてはならない情報を最小限におさえる。
【構成】　ライトーアヘッドログープロトコールを用いてアンドゥーログの最適化を実施するために、書込むブロックがコミットしていないデータを含むかどうかを判定し１０１０、含んでいる場合はアンドゥーバッファーをアンドゥーログに書き込み、リドゥーバッファーをリドゥーログに書き込み１０２０、その後又はブロックがコミットしてないデータを含まない場合には１０１０、ブロックを永久記憶装置に書込む１０３０。



1

【特許請求の範囲】

【請求項1】　複数のノードとセクションに分割された非揮発性記憶媒質とを有するデータ処理システムであって、複数のノードがトランザクションによってセクションに変化し、各トランザクションは少なくとも1つのノードによって少なくとも1つのセクションになされた一連の変化から成り、各トランザクションはもしトランザクション及びトランザクションの完了の指示によって行われた変化の記録が記憶媒質に確実に記憶されるならば委任され、さもないと委任されないデータ処理システムにおいて、複数のノードのうちの最初の1つは、

少なくとも1つのセクションのコピーを保持するメモリと、

メモリに接続されて、メモリ内の少なくとも1つのセクションのコピーに変化する処理手段と、

処理手段によってメモリ内の少なくとも1つのセクションのコピーになされる変化の順次リストを含む少なくとも1つのアンドゥーバッファとを有し、各アンドゥーバッファは最初のノードによって異なった委任されてないトランザクションになされた変化に対応し、

対応するノード内の処理手段によってメモリ内の少なくとも1つのセクションのコピーになされた変化の順次リストを含むリドゥーバッファと、

メモリに接続されて、少なくとも1つのセクションのコピーを記憶媒質に格納する記憶手段と、

アンドゥーバッファ及びリドゥーバッファに接続されて、アンドゥーバッファ及びリドゥーバッファの一部を記憶媒質に選択的に格納して委任されてないトランザクションのすべての変化の効果を除去しかつ委任されたトランザクションのすべての変化の効果を再形成するのを確保するログ管理手段とを有することを特徴とするデータ処理システム。

【請求項2】　請求項1記載のノードにおいて、ログ管理手段は記憶手段が対応する委任してないトランザクションからの変化を格納する前に記憶媒質内の復元バッファの内容を格納する手段を有することを特徴とするノード。

【請求項3】　請求項1記載の最初のノードにおいて、ログ管理手段は反復バッファに記録された最初のトランザクションに対する変化のすべてと最初のトランザクションの完了の指示とを記憶媒質に確実に格納し、それによって最初のトランザクションを委任させる手段を有することを特徴とするノード。

【発明の詳細な説明】

【発明の背景】本発明は一般に共用ディスクシステムにおけるクラッシュからの回復の分野、特にかかる回復におけるログの使用に関する。総てのコンピュータはクラッシュするとデータを失うことがある。あるシステムの中には故障或いははクラッシュによりデータを失う可能性が特に高いものがある。これはそれらのシステムがディスク及びプロセッサメモリ間で大量のデータを伝送するためである。データを失うことに共通した理由とは非持久型記憶システム（例えばプロセッサメモリ）から持久型記憶システム（例えばディスク）へデータを送る際の不完全な伝送である。クラッシュ時に処理が行なわれるとしばしば不完全な伝送が生じる。一般に処理は2つの記憶システム間の一連の記憶（或いは変更）の伝送を含んでいる。データの紛失及びその回復をアドレスする際に重要な概念は処理を「実行」するという考えである。処理はその総ての効果が持久型記憶機構で安定していることをある程度保証されている時に実行されるのである。処理が実行される前にクラッシュが生じた場合、回復に必要な工程は処理後にクラッシュが生じた場合の回復に必要なものとことなる。回復とは、知られた所望の時点で完全なシステムを再開できるようにデータベースを訂正する工程である。必要とされる回復の種類は当然ながらデータの紛失理由による。コンピュータのシステムがクラッシュすると、その回復手段はクラッシュシステムの例えばディスク等の持久型記憶機構を前回実行された処理により生じたものと一致する状態まで回復できる必要がある。持久型記憶機構がクラッシュ（媒体故障と言う）すると、その回復には貯蔵されたデータをディスクに再生する必要がある。データベースシステムを回復するための多くの手段はログを使用している。ログは少なくともデータベースの場合、そのデータベースにどんな変更が行われたのか、またそれらの変更はどのような順序で行われたのかを示す時刻順のアクションのリストに過ぎない。従って、ログはコンピュータシステムにデータベースを知られた所望の状態にするものであり、その後データベースはリドゥー（REDO）或いはアンドゥー（UNDO）の変更に使用できる。リドゥーは再実行を、アンドゥーは無効を意味する。しかし、ノードと呼ばれる数多くのコンピュータシステムが共用ディスクの集合をアクセスするシステム配置においてはログは管理しにくい。この構成のタイプは「クラスター」或いは共用ディスクシステムと呼ばれる。このようなシステムにおいてノードにデータをアクセスさせるシステムをデータ共用システムと呼ぶ。データ共用システムはデータの送信を行い、これによりデータブロック自身がディスクから要求するコンピュータに送られる。反対に「区分」システムとしてより広く知られる機能送信システムはデータの区分の「サーバー」として指定されたコンピュータにオペレーションの集合を送信する。その後、サーバーはオペレーションを実行してその結果を要求する側に返送する。区分システムでは単一ノード或いは集中システムのようにデータの各部は1ノード以下のローカルメモリの中に属することができる。また、区分システム及び集中システムは両方共単一のログに記憶されたアクションしか必要としない。ここで重要なのはデータの回復は1つのログの内容だけに基づき行なえることであ

る。一方、分配されたデータの送信システムは集中化されず、同じデータが多数のノードのローカルメモリ内に属してそれらのノードから更新できる。これにより同じデータのために多数のノードロギングアクションが生じる。同じデータのために多数のログを有するアクションの問題を避けるために、データ送信システムは、データの回復情報を記録する責任のある単一のログへデータのためのログの記録を返送することを要求する。しかし、このような遠隔ロギングには別々に新たなシステム源が要求される。何故なら入出力のほかに必要なログの記録を有する別のメッセージをログに書き込むためである。更にロギングコンピュータから受信通知を待つことにより遅れが大きくなり得る。これにより応答時間が増加するだけでなく、ユーザーが同じデータベースへ同時にアクセスできいるという能力を減少させることにもなる。別の方法としては代わる代わる書き込む共通のログを同時に使用することである。しかし、これも調整のための別のメッセージを必要とするため費用がかかり過ぎる。これらの問題点はアドレスにとって重要である。何故ならおうおうにしてデータ共用システムのほうが区分システムより好ましいからである。たとえばデータ共用システムはワークステーションに延長された期間データを貯蔵させることができ、これによりデータの局所処理を高い水準で実行できるため、データ共用システムはワークステーション及びエンジニアリング設計アプリケーションにとって重要である。更に、データ共用システムは多数のノードがデータを同時にアクセスし、いくつかの局所データ自体を管理し、また他のデータを他のホストコンピュータ及びワークステーションと共用できるために、本来的に耐故障性及びロードバランス性を有している。従って、本発明の目的はリドゥーレコードからアンドゥー情報を除去することによりリドゥーログの管理を容易にすることである。また本発明の他の目的はトランザクション実行時にアンドゥー情報を放棄することによりアンドゥー情報の管理をより容易にすることである。更に本発明の他の目的は、クラッシュ或いは故障が発生した際にアンドゥートランザクションに記録させなくてはならない情報を最小限におさえることである。

II．発明の要約

本発明は実行していないトランザクションの総ての変更を除去でき、実行されたトランザクションからの変更を再生でき、またアンドゥーログへのアンドゥーバッファの記録量を最小限に抑えることができるようにリドゥー及びアンドゥーバッファから充分な情報を維持することにより、先行技術の問題点を回避するものである。更に、アクションは行われないため、トランザクションにおけるアクションのカウントを保持することにより効率を維持することができる。詳細には、複数のノード、及びセクションごとに分割された持久型記憶媒体を有するデータ処理システムにおいて、前記複数のノードはトランザクションにより前記セクションに対して変更を行い、前記各トランザクションは少なくとも1つのノードにより、少なくとも1つのセクションに対して行なう一連の変更を有し、前記各トランザクションは、該トランザクションにより行われる変更のレコード及び該トランザクションの完了の表示が記憶媒体に確実に記憶された場合に実行され、確実に記憶されない場合には実行されず、前記複数のノードの最初の1つは、少なくとも1つのセクションのコピーを保持するメモリと；該メモリに連結されて、そのメモリ内の少なくとも1つのセクションのコピーに対して変更を行なう処理手段と；前記メモリ内の少なくとも1つのコピーに対して前記処理手段により行われた変更の順次リストを有する少なくとも1つのアンドゥーバッファと；該各アンドゥーバッファは最初のノードにより該ノード及び各アンドゥーバッファのための実行されていない異なるトランザクションに対して行われた変更に対応し、対応するノード内の処理手段によりメモリ内の少なくとも1つのセクションのコピーに対して行われた変更の順次リストを有するレコードバッファと；前記メモリに連結され、少なくとも1つのセクションのコピーを前記記憶媒体に戻して記憶するための記憶手段と；実行されていないトランザクションの全変更の効果を除去できること、及び実行されたトランザクションの全変更の再生できることを確実にするために、前記アンドゥーバッファ及びリドゥーバッファの部分を前記記憶媒体に選択的に記憶する、アンドゥーバッファ及びリドゥーバッファに連結された、ログ管理手段と；よりなることを特徴とする。本願の一部である添付図面は本発明の好適な実施例を示し、本明細書と共に本発明の原理を説明するものである。

【好適な実施例の説明】本発明の好適な実施例を以下に詳述する。その具体例を添付図面に図示する。

A．システムの構成要素

システム100は本発明を実施するのに使用できる記憶システムの一例である。システム100はノード110、120及び130を有し、これらは総て共用ディスクシステム140をアクセスする。ノード110、120及び130は後述の記憶及び回復ルーティーンを実行するためにそれぞれプロセッサ113、123及び133を有する。ノード110、120及び130は少なくとも2つの機能を提供するためにそれぞれメモリ118、128及び138を更に有している。これらの機能のうち一方は対応するプロセッサのための局所メモリとして作動し、他方は前記ディスクシステム140と交換されてデータを保持するものである。データ交換に使用されるメモリの部分はキャッシュと呼ばれる。キャッシュは一般に非持久型記憶システムである。共用ディスクシステム140もたとえ永続的記憶システムの一部或いは総てがクラッシュしても、記憶内容は永続であると考えられる持久型記憶システムを指す。この持久型システ

ムは伝統的に磁気ディスクシステムを採用しているが、永続的記憶システムは光学式ディスク或いは磁気テープシステムを使用することもできる。また、本発明を実施するのに使用される永続的記憶システムは図１に示されるアーキテクチャー（コンピュータの設計仕様）にのみ限定されるものではない。例えば、この永続的記憶システムはそれぞれ異なるノードに結合されたいくつかのディスクを使用することもでき、それらノードはいくつかのタイプのネットワークに連結されても良い。永続的記憶システムの他の部分はアーカイブ（ａｒｃｈｉｖｅ）記憶システムと呼ばれるバックアップシステム１５０である。アーカイブ記憶システムという用語は永続的記憶システム内のデータが読み取り不能になった際に永続的記憶システムの内容を復元させる情報のために使用される記憶システムに言及するのに一般に使われている。例えば共用ディスクシステム１４０が媒体に障害を起こした時、そのシステム１４０を復元するためにテープシステム１５０を使用することができる。アーカイブ記憶システムは通常磁気テープシステムを使用しているが、磁気或いは光学式ディスクシステムを使用することもできる。システム１００のデータは通常ブロックに記憶され、それは該システムの回復可能な目的である。一般にブロックはそれらがあるノードのキャッシュ内にある時のみ作動できる。図２はディスク２００の一部のブロック２１０、２２０及び２３０の一例を示したものである。一般に１つのブロックは永続的記憶システムの整数のページ数を有している。例えば、図２においてブロック２１０はページ２１２、２１４、２１６及び２１８を有している。

B．ログ

上述の通り、データベースシステムの殆どは回復を目的としたログを使用している。ログは通常永続的記憶システム内に記憶される。ノードが永続的記憶システムを更新する時、ノードはそのノードのキャッシュ内のバッファ内の更新を記述するログレコードを記憶する。本発明の好適な実施例では永続的記憶システムにおいて３つのタイプのログを想定しているが、各ノードのキャッシュには２つのタイプのバッファしかない。前記ログとはリドゥーログ（ＲＬＯＧ）、アンドゥーログ（ＲＬＯＧ）及びアーカイブログ（ＡＬＯＧ）である。また、前記バッファとはリドゥーバッファ及びアンドゥーバッファである。ＲＬＯＧの例を図３に、ＵＬＯＧの例を図４、またＡＬＯＧの例を図５に示す。リドゥーバッファの構成はＲＬＯＧに類似しており、アンドゥーバッファの構成はＵＬＯＧに類似している。ログの順序番号（ＬＳＮ）はログ内のレコードのアドレス或いは相対的位置である。各ログはそのログ内のレコードにＬＳＮを示す。

1．　RLOG（リドゥーログ）

図３に示されるようにＲＬＯＧ３００は変更に関する情報を記録するために使用される順次ファイルの好適な実施例であり、その情報はそれらの変更が示される間に行われる特定のオペレーションを繰り返すことを許容するものである。一般に、これらのオペレーションは一度ブロックが記録されたアクションの実行された状態まで復元されると、回復計画の間反復させる必要がある。図３に示されるように、ＲＬＯＧ３００はレコード３１０、３０２及び３１０を有し、各レコードは幾つかの属性を有する。ＴＹＰＥ属性３２０は対応するＲＬＯＧレコードのタイプを識別する。ＲＬＯＧレコードの異なるタイプの例としてはリドゥーレコード、補正ログレコード及び実行に関するレコードである。これらのレコードについて以下に説明する。ＴＩＤ属性３２５はカレントレコード（現在レコード）に関連するトランザクションのための唯一の識別名である。この属性は現在のＲＬＯＧレコードに対応するＵＬＯＧのレコードを探すための補助に使用される。ＢＳＩ属性３００は状態前識別名（ｂｅｆｏｒｅ　ｓｔａｔｅ　ｉｄｅｎｔｉｆｉｅｒ）である。この識別名を以下に詳述する。簡単に言えば、このＢＳＩは対応するトランザクションにより修正変更される前のブロックのバージョンのための状態の識別名の値を示す。ＢＩＤ属性３３５はＲＬＯＧレコードに対応する更新により修正変更されたブロックを識別する。ＲＥＤＯ　ＤＡＴＡＭ属性３４０は対応するアクションの性質を記述し、そのアクションを繰り返すための充分な情報を提供する。「更新」という用語は本明細書の中で広義に、且つ「アクション」という用語と同じ意味に使用される。アクションとは厳格に言えば、レコードの変更だけではなく、レコードの挿入及び削除、またブロックの割り振り及び開放をも含む。ＬＳＮ属性３４５は唯一ＲＬＯＧ３００のカレントレコードを識別する。後述するように、ＬＳＮバッファ３４５は本実施例ではレドスキャン及びＲＬＯＧチェックポイントを制御するのに使用される。ＬＳＮ３４５は本実施例ではＲＬＯＧレコード或いはブロックのいずれにも記憶されない。それはＲＬＯＧ内のレコードの位置から本来固有のものだからである。本発明の目的は各ノードに他のノードとは可能な限り独立してその回復を管理させることである。このために、別個のＲＬＯＧを各ノードに関連させる。好適な実施例におけるノードとＲＬＯＧとの関連は各ノードに異なるＲＬＯＧを使用する方法を用いている。別の方法として、ノードにＲＬＯＧを共用させるか、或いは各ノードに多数のＲＬＯＧを与えることもできる。しかし、ＲＬＯＧが１つのノード専用である場合には、そのとＲＬＯＧをほかのＲＬＯＧやノードに同等に使用するためのメッセージを含む同期化は必要ない。

2．　ULOG（アンドゥーログ）

図４において、ＵＬＯＧ４００は、ブロックのオペレーションを正しく取り消す情報を記録するのに使用される順次ファイルの好適な実施例である。ＵＬＯＧ４００はブロックをトランザクションが始まった時の状態に復元

させるのに使用される。ＲＬＯＧとは異なり、ＵＬＯＧ及びアクシヨンバッファはそれぞれ異なるトランザクションに関連する。従って、ＵＬＯＧ及びそれらの対応するバッファはトランザクションが実行されると消失し、新たなトランザクションが開始すると新たなＵＬＯＧが出現する。他の可能性も存在する。ＵＬＯＧ４００はレコード４１０、４０２及び４１０を有し、各レコードは２つのフィールドを有する。ＢＩＤフィールド４２０はそのレコードに記録されたトランザクションにより修正変更されたブロックを識別する。ＵＮＤＯ－ＤＡＴＡフィールド４３０は更新の性質を記述し、更新されたものを元に戻すのに充分な情報を提供する。ＲＬＳＮフィールド４４０は或るアクションが取消された時のために同じアクシヨンを記述するＲＬＯＧレコードを識別する。この属性は各ＵＬＯＧを別々に識別する機能を提供するためのものである。

３．　ＡＬＯＧ（アーチブログ）

図５においてＡＬＯＧ５００は図１の共用ディスクシステム１４００が故障した時等の媒体の回復に充分な時間レドログの記録を記憶させるのに使用される順次ファイルの好適な手段である。ＲＬＯＧバッファはＡＬＯＧ５００が発する情報源であり、従って、ＡＬＯＧ５００はＲＬＯＧ３００と同じ属性を有する。ＡＬＯＧは好ましくは対応するＲＬＯＧの切断部分から形成される。切断部分はブロックの永続的記憶システムのバージョンをカレントバージョンにする必要のない部分である。しかし、ブロックの永続的記憶システムバージョンが入手不可能になった場合や、アーチブ記憶システムのブロックのバージョンから回復させることが必要になった場合、このＲＬＯＧの切断部分のレコードはまだ必要である。ＲＬＯＧ３００と同様にＡＬＯＧ５００はレコード５０１、５０２及び５０３を有する。ＴＹＰＥ５２０、ＴＩＤ５２５、ＢＳＩ５３０、ＢＩＤ５３５及びＲＥＤＯ－ＤＡＴＡ５４０の属性は同じ名称のＲＬＯＧ３００における属性と同様に機能する。ＬＳＮ５４５はＲＬＯＧ３００のためのＬＳＮ３４５と同じくＡＬＯＧのレコードを識別する。

Ｃ．状態識別名（及びライトーアヘッドログプロトコル）

ログベースシステムではログレコードは、記録されたブロックの状態がログレコードに指定された更新に適切である場合にのみブロックに出力される。従って、レドの訂正に充分な条件はブロックがオリジナルのアクションの遂行された時と同じ状態にある時に、記憶されたトランザクションをブロックに出力することである。オリジナルアクションが正しければ、やり直されたアクシヨンも正しいであろう。ブロックの状態の全体の内容をログに記録させるのは難しく、実用的ではない。従って、ブロックの状態のために代理の値、或いは識別名が作られる。本実施例で使用されるその識別名は状態識別名、即ち、ＳＩである。ＳＩは各ブロックのための唯一の値を有する。その値はブロックのオペレーションの実行前、或いは後等のように特定の時間におけるブロックの状態を識別する。ＳＩは完全な状態よりもはるかに小さいため、必要な時に完全な状態が再現できる限り、完全な状態の代用として安価に使用できる。ＳＩはブロック内の区画状態識別名、即ち、ＤＳＩと呼ばれる特定の値を記憶することにより「区画」される。このＤＳＩはそれが含まれるブロックの状態を示す。状態の再生は回復の間に永続的記憶システムに記憶された全体のブロックをアクセスすると共にそのブロックのＤＳＩを示すことにより行なわれる。このブロックの状態は適切なログアクションを行なうことにより、後述のように更新される。ＡＬＯＧを用いた媒体の回復のための類似した方法を次に述べる。ログレコードがブロックに出力されたか否かを知ることにより、ログアクションがどのような状態を出力したかをログレコードにより判断できる。本発明によれば、ブロックのＤＳＩはログレコードをそのブロックにいつ出力したかを判断するのに使用される。集中或いは区分システムでは、アクションをやり直すために単一のログのレコードの物理順序が用いられる。即ち、あるブロックのアクションＢがそのブロックのアクションＡの直ぐ後に続く場合、そのアクションＢはアクションＡに再生されたブロック状態に適用される。従って、アクションＡがやり直されれば、ブロックに適用される次のブロックレコードはアクションＢである。集中、或いは区分システムのような単一のログシステムは通常ブロックの状態を識別するのにＤＳＩとしてＬＳＮを使用している。ブロックのためのＤＳＩとして機能するＬＳＮはブロック内で効果を反映させるべくログ順序における前回のレコードを識別する。かかるシステムでは、ログレコードのＬＳＮは記憶されたアクションの後のブロック状態を識別する「状態後識別名」、即ちＡＳＩとして機能する。これは後述するようにログレコード内で本発明に使用されたＢＳＩ（状態前識別名）と対象的である。ＤＳＩを更新して次のオペレーションに備えるため、ログレコードを出力した後のブロックのためのＡＳＩも決定できる必要がある。記憶可能であるにも関わらずＡＳＩをログレコードに記憶させなくても良いように、そのＡＳＩをＢＳＩ等のログレコードから獲得できるのが有効である。しかし、そのような獲得は回復や正常なオペレーション時に使えるものでなくてはならない。好ましくは、ＳＩは単調に増加し、ゼロから始まる整数の組等の知られた順序である。この技法では、ＡＳＩは常にＢＳＩより１だけ多い。更新されたブロックを共用ディスクシステム１４０等の永続的記憶システムに記憶する場合、ライトーアヘッドログ（ＷＡＬ）プロトコルが使用される。ＷＡＬプロトコルはレド及びアンドゥーバージョンをブロックの前の共用ディスクシステム１４０内のログに書き込むことを要求する。これはアクションの反

9

復、或いは取消に必要な情報がデータの永続的コピーを変更する前に安定して記憶するためである。WALプロトコルが理解されずに、ブロックの前回の更新のためのログレギュレータ以前にブロックが永続的記憶システムに書き込まれようとしても、一定の条件下では回復はできない。例えば、1つのノードで更新することにより、実行されない更新を含むブロックは永続的記憶システムに書き込まれる。そのブロックに対する前回の更新がノードのRLOGに記憶されておらず、また第2のノードの他のトランザクションがブロックを更新して実行した場合、そのブロックのDSIは増加する。その第2のトランザクションのための実行時に、他のトランザクションのための記憶されたアクションが第2のノードのためのRLOGに送られる。何故なら、2回目の更新は別のノードにより生じたものではあるが、第2のトランザクションのためのログレコードの書き込みからは、オリジナルノードの実行されていないトランザクションのためのログレコードが書き込まれるのを保証しないためである。オリジナルノードがクラッシュし、また実行されていないトランザクションのためのログレコードがRLOGに書き込まれていない場合、ブロックのためのASI－BSI順序にギャップが生じる。例えばディスクの故障等により永続的記憶システム内のブロックが入手不可能になった場合、後述のALOGマージが知られているSIのギャップのない順序を要求するため回復は失敗するであろう。従って、WALプロトコルは記憶されたアクションの破壊されない順序のために必要な条件である。それはブロックの更新に関しても充分な条件である。ブロックが1つのノードのキャッシュから別のものに移った場合、WALプロトコルは更新前のRLOGレコードを、トランザクションを実行することにより更新されるブロックに強制する。「強制」とはノードキャッシュ、或いはバージョン内のレコードを永続的記憶システムに安定して記憶することを確実にするという意味である。永続的記憶システムに書き込むことにより、WALプロトコルは前回更新されたログレコードからカレントブロックまで、オリジナルノードのRLOG内の総てのレコードを書き込むことを強制する。

D．新たなブロックの割り振り

正常なディスク記憶管理ルーティーン等の間にブロックが開放され、後の段階で更に使用するために再度割り振られた場合、このアクションは特殊ではない状態の識別名になるため、DSIをゼロに設定してはならない。DSIをゼロに設定すると、いくつかのログレコードが出現してブロックに適用される可能性がある。何故ならそれらは同じSIを有しているからである。正確なログレコードを判断するために新たな情報が必要であろう。従って前回の割り振りに使用されたDSIの連番は新たな割り振りにおいて区切れずに保護しなければならない。新たに割り振られたブロックのためのBSIは開放されたブロックのASIであるのが好ましい。区切れないSIの連番を達成するのに簡単な1つの方法は、DSIをフリーオペレーションの結果としてブロック内に記憶することである。ブロックが再度割り振られると、それは恐らく永続的記憶システムから読み取られ、正常にDSIの増加が続く。これは丁度更新オペレーションのように割り振り及び開放を扱う。この解決策の1つの問題点はブロックを使用する前に、新たに割り振られたブロックを読み取る必要があることである。しかし、最小の入出力アクティビティーでスペース管理を効果的に行なうためには、「割り振り前の読み込みによりペナルティー」を回避するのが望ましい。本発明は割り振られていないブロック全てにDSIを書き込まないことにより効率を高めるものである。以前割り振られていなかったブロックのために初期のDSIは常にゼロに設定される。そして割り振り解除されたブロックのDSIだけが記憶される。これらのDSIは永続型記憶システム内のフリースペースの簿記のためのシステムにより既に保持されているレコードを使用して記憶される。通常このような簿記の情報はスペース管理でブロックの集合内に記録される。このスプレー管理の情報と共に、割り振り解除された各ブロックのための初期SIを記憶することにより、その初期SIをブロック内に記憶する必要がなくなり、従って割り振り前の読み取りペナルティーを避けることができる。再割り振りでは、「割り振り」オペレーションのためのBSIは以前の「解放」ブロックの初期SIになる。勿論、この処理を正確に作動させるには、スペース管理の情報を含むブロックを永続型記憶システムに定期的に書き込まなければならず、1つのノードは解放されたブロックの存在が簿記を介して明らかになるまで、他のノードにより解放されたブロックを割り振ってはならない。従って、解放されたブロックのために初期SIを維持することにより、フリースペース簿記の情報の新たな読み取り或いは書き込みの必要が生じることはない。フリーブロックのためのSI情報を追加すればこのシステムに必要なスペース管理情報の量は増加するが、システムの効率がそれ程低下しない2つの理由がある。第1の理由は、殆どのフリースペースは「以前に割り振られてはいない」（never before allocated）という特徴を有しており、従って既にゼロの初期SIを有している。第2の理由は、データベースは通常成長するものであるので、以前使用されたフリースペースは殆どのデータでは小さい。初期SIは再度割り振られたブロックのためだけに個別に記憶されるため、SIの増加した記憶量は小さい。別の方法として、以前割り振られていないブロックを再度割り振られたものと区別することができる。再割り振りされたブロックのSIはそれらが割り振られる時永続型記憶システムから読み取られる。しかし、上述の理由によりペナルティーの程度は軽いものの、これにより割り振り前の読

11

み取りペナルティーが生じる。

E．回復

1．　ブロックバージョン

回復においてログをいかにして使用できるかを理解するために、クラッシュ後に利用できる、異なるブロックバージョンを把握する必要がある。これらのバージョンは、利用できるカレントバージョンを作成するのにどれだけ多くのログが必要かに基づく記憶された更新の量として表現することができる。これはどのくらい広い或いは局所化された回復アクティビティーであるかについて明らかにインパクトを有する。回復のために、3種類のブロックがある。ブロックで実行された全ての更新がバージョン内に反映されていれば、そのブロックのバージョンが「カレント」バージョンである。故障後のカレントバージョンを有するブロックはレド回復を必要としない。しかし、予測できないシステムの故障に対処するとき、更新するたびにキャッシュを永続型記憶システムに全て書き込まなければ全てのブロックがカレントであることを保証できない。これは高価でありまれにしか行われない。ただ一つのノードのログが、まだブロックに送られていない更新を有している場合、ブロックのバージョンは「ワン－ログ」（ｏｎｅ－ｌｏｇ）である。故障が起きても多くても1つのノードが回復に使用されれば良い。これは回復の間に、調整が高価についたり、また新たに費用がかかる可能性を回避するので望ましい。1つ以上のノードのログが、まだ適用されていない更新を有することができる場合、ブロックのバージョンはＮ－ログである。ワン－ログブロックよりもＮ－ログの方が一般に回復がより難しいが、媒体メモリを提供する時にブロックが常にワン－ログであることを確実にすることは実用的ではない。何故なら、これはブロックがノードを変えるたびに、ブロックをアーカイブ記憶システムに書き込み工程を含むからである。

2．　レド回復

システムのクラッシュ時に媒体とは対照的に、いくつかのブロックが不注意によりＮ－ログになることがある。しかし、本発明の好適な実施例では、全てのブロックが、システムのクラッシュからの回復のためのワン－ログブロックであることを保障している。Ｎ－ログブロックは回復のためのノード間で複雑な調整を必要とすることがあるため、これは利点である。更新は正常なシステムの作動中に、分散された同時制御により最初に連続して行われたため、上述のような調整は可能ではあるが、そのような同時制御は回復中には回避すべきであるオーバヘッドを要求する。全てのブースタは、あるキャッシュから他のものへ移動する前に永続型記憶システムに書き込むよう「ダーティー」ブロックに要求することにより、レド回復に関してワン－レグになることが保障される。ダーティーブロックとは、そのブロックが永続型記憶システムから読み取られているため、キャッシュにおけるバージョンが更新されたものである。この規則に従えば、要求するノードは、ブロックが新しいノードのキャッシュに入る時はいつでもクリーンブロックを得ることができる。更に、回復の間、ブロックを変更する前回のノードのログのレコードだけをそのブロックに適用すれば良い。他のノード他の全てのアクションは永続型記憶システム内のブロックの状態において既に獲得されている。従って、全てのブロックはリドゥーレコードに対してワン－ログになり、そのためレグ回復は分散された同時制御を必要としない。この技法は多数のログはブロックのためのレコードを決して含まないという意味ではない。この技法は1つだけのノードのレコードが永続型記憶システム内のブロックのバージョンに適用できることを確実にするだけである。更に、システムのクラッシュのためにワン－ログリドゥーの回復を用いることが考えられるが、媒体の回復を実行するためには、ブロックがキャッシュ間を移動するためだけに各ブロックがアーカイブ記憶システムに書き込まれるのを避けるために多数のログのリドゥーアクションを適用しなければならないかも知れない。従って、特定の環境では全てのログにわたって記憶されたアクションにＮ－ログブロックのための回復を提供するよう命令する必要が依然としてある。しかし、ＳＩ順序のため、これは実行できる。図6は上述のＲＬＯＧ及びＳＩを使用したリドゥーオペレーションのための基本的工程のフローチャート６００を示す。このフローチャート６００で示されるレドオペレーションは単一のブロックに適用される単一のＲＬＯＧレコードを使用して単一ノードにより行われる。先ず、ログレコードにより識別されたブロックの最も新しいバージョンはＰＳ（工程６１０）から呼び戻される。その呼び戻されたブロックに記憶されていたＤＳＩがログレコード（工程６００）内に記憶されたＢＳＩと等しければ、そのログレコードに示されたアクションがそのブロックに適用され、ＤＳＩが増加して前記ブロックの新たな状態を反映する（工程６３０）。前記ＤＳＩが前記ＢＳＩと等しくない場合には、その更新はブロックには適用されない。ＢＳＩ及びＡＳＩは回復時に判断されるため、図6に関して述べられたリドゥーオペレーションは可能である。従って、各ログのために、どのログレコードをやり直さなければならないのかを判断でき、この判断は他のログの内容と独立させて行うことができる。ブロックＤＳＩ及びログレコードＢＳＩ間で行わなくてはならない唯一の比較は同格性の比較である。図6に関して記述されたレドオペレーションはシステムのクラッシュからの回復に使用することができる。クラッシュからの回復の手順を図7のフローチャート７００に示す。単一のノードが他のノードとは独立してこのクラッシュからの回復処理を実行できる。最初の工程は最新のチェックポイントにより示された最初のＲＬＯＧレコードをノードに読み込ませることである（工程７１０）。後述す

13

るように、チェックポイントは適用しなければならない最も古い更新に対応するレコードを含むＲＬＯＧにおけるポイントを示す。その後、図６に示されるリドゥーオペレーションはそのログレコードに特定されたアクションをそのログレコードに識別されたブロックに適用するかどうかを判断する（工程７２０）。リドゥーオペレーション終了後にそれ以上レコードがなければ、クラッシュの回復は完了する。反対に他にもレコードがある場合には、次のレコードがＲＬＯＧから呼び戻され（工程７４０）、図６に示されるリドゥーオペレーションが繰り返される（工程７２０）。ログレコードに関連したＳＩがＡＳＩを単調に増加する場合、ログレコードをある状態のブロックに適用すべきか否かのテストは、このＡＳＩがそのブロックのＤＳＩより大きい最初のものであるか否かのテストである。しかし、これはワン－ログの回復だけにとってみれば充分である。何故なら、その場合、ワン－ログだけがブロック内のＤＳＩよりも大きなＡＳＩを有するレコードを持っているからである。しかし、本発明の好適な実施例では、各ログレコードは記憶されたアクションを実行する前と正確に同一のブロック状態を含んでいる。上述のように、これが“ｂｅｆｏｔｅ　ｓｔａｔｅ　ｉｄｅｎｔｉｆｉｅｒ”（状態識別名の前）、即ち、ＢＳＩである。

３．　媒体回復のための多数のログレコ

媒体の回復はクラッシュの回復と同じ性格を多く有している。例えば、ログレコードが適用される、安定して記憶されたバージョンが必要である。また、重要な違いもある。第１に、ＡＬＯＧレコードが適用されるブロックの安定したバージョンは前回のアーチブ記憶システムに送られたバージョンである。媒体の回復はＮ－ログである。何故ならそれらはアーカイブ記憶システムからブロックを復元し、また上述のようにブロックはそれらがキャッシュ間を移動する度にアーチズ記憶システムに書込まれないからである。従って、システムのクラッシュのためのＮ－ログ回復を避けるためにブロックを書き込んで記憶するという技法は媒体の回復には使用できない。媒体の回復を管理するにはＡＬＯＧを組合せなければ困難である。ＡＬＯＧと組合せない場合、回復は適用可能なログレコードのための一定の検索を必要とする。ＡＬＯＧと組合せると、ＢＳＩの使用に際して大きな利点がある。図８は多数のＡＬＯＧのマージャーを有するＮ－ログ媒体の回復のための処理８００を示している。組合せは全てのログレコードの中の命令の統計に基づいてではなく、同じブロックのためのログレコードの中の命令により生じた部分的命令に基づいて行われる。時々、対応するそれぞれのブロックに適用できる。アクションのレコードを有する多数のＡＬＯＧがある。処理８００の記憶から明らかなように、媒体の回復時にこれらのアクションのどれを最初に適用するかは重要ではない。多数のＡＬＯＧを組合せて、単一のパス内のアーカイブ記憶

14

システムにおけるバックアップデータベースに適用する方が速くて効果的である。これはＳＩが正しく命令された場合、行うことができる。上述の通り、ＳＩは知られた順序で命令され、また本発明の好適な実施例が単調に増加するＳＩを使用しているのはこのためである。いずれのＢＬＯＧから開始しても最初のログレコードはアクセスされる（工程８１０）。その後、ブロックＩＤ及びＢＳＩはそのレコードから抽出される（工程８２０）。次にブロックＩＤにより識別されたブロックが取り出される（工程８３０）。一度、識別されたブロックが取り出されると、そのＤＳＩは読み込まれ、ＡＬＯＧレコードのＢＳＩと比較される（工程８４０）。ここで、ＡＬＯＧのレコードのＢＳＩがブロックのＤＳＩより小さければ、記憶されたアクションは既にブロックに盛り込まれているということであるためそのレコードは無視され、従ってレドは要求されない。また、ＡＬＯＧレコードのＢＳＩがブロックのＤＳＩと等しければ、記憶されたアクションはブロックに適用されてやり直される（工程８５０）。これは、ＳＩが等しいということは記憶されたアクションがブロックのカレントバージョンに適用されるという意味であるからである。その後ブロックのＤＳＣは増加する（工程８６０）。これは記憶されたアクションのアプリケーションがブロックの新しい（より最新の）バージョンを作り出したことを示す。ここでＡＬＯＧレコードのＢＳＩがブロックのＤＳＩより大きい場合、ログレコードに対応するアクションを適用するのには適切な時ではなく、代わりに他のＡＬＯＧに記録されたアクションを適用するのに適切な時である。従って、このＡＬＯＧの読み込みは休止しなければならず、他のＡＬＯＧの読み込みが開始される（工程８７０）。他のＡＬＯＧが既に休止していた場合には（工程８８０）。コントロール工程８２０に移しそのログが休止した時にカレントであったログレコードのブロックＩＤ及びＢＳＩを抽出する。また、ログが休止していなかった場合には、コントロールはそれがあたかも最初のＡＬＯＧであったとして処理を続ける。結局、これらの工程、即ち他のＡＬＯＧが前もって休止していなかった場合、ＡＬＯＧレコードのいずれかを残すか否かの判断が行われる（工程８９０）。残すのであれば次のレコードは取り出された（工程８１０）。残さないのであれば工程８００を終了する。ＡＬＯＧを休止する場合、カレントブロックより先行するブロックのレコードを有する他の少なくとも１つのＡＬＯＧがなくてはならない。特機ログレコードを有する休止したＡＬＯＧは単に、その最初のアイテム（配列された順序で）が他の入力ストリーム（即ちＡＬＯＧ）におけるアイテムよりも後で比較する入力ストリームと見做される。処理は他のＡＬＯＧを用いて続行される。ＢＳＩは他のＡＬＯＧに介入するアクションなしにはブロックのＤＳＩよりも大きくはないため、休止したＡＬＯＧのカレントレコードは後の段階で

15

ブロックに適用できなくてはならない。これが生じた時、休止していたＡＬＯＧは休止が解除される。アクションはブロックのためのＳＩの配列に合致した配列で最初に行われたため、ＡＬＯＧの全てが同時に休止することはない。従って、ＡＬＯＧのマージは常に可能である。

4．　リドゥー管理

a．安全点の決定

本発明に関し、リドゥー回復をよりいっそう効率的にするのに、多くのチェックポイント設定テクニックを用いることができる。例えば、ダーティブロックテーブルを作成して、回復管理情報を各ダーティブロックに対応づけすることができる。この情報は、ＲＬＯＧ管理に、従ってＡＬＯＧに、二つの重要な機能を提供するものである。第一に、回復管理情報は、ＲＬＯＧの走査及び切捨を支配する「安全点」の決定に用いられる。第二に、回復管理情報は、ポテンシャル取消ログについてだけでなく、ＲＬＯＧについてもＷＡＬプロトコルを実施する（ｅｎｆｏｒｃｅ）のに使用できる。安全点の決定は、リドゥー回復を行うために、どれだけ多くのＲＬＯＧが走査を必要とするかを決定するのに重要である。このリドゥー走査に関するＲＬＯＧの開始点が、「安全点」と呼ばれる。安全点は、二つの意味で「安全」である。第１にリドゥー回復は、安全点に先行するレコードを安全に無視することができ、これはそれらのレコードが、既に全て永久記憶装置におけるブロックのバージョンに含まれているからである。第二に、「無視した」レコードは、もはや必要がないので、それらのレコードをＲＬＯＧから切捨てることができる。この第二の特徴は、アンドゥー／リドゥーログに関しては正しくない。例えば、アンドゥーレコードが生成する長いトランザクションがあったならば、アンドゥーレコードにおける動作が、永久記憶装置に書込まれているリドゥーレコードにおける動作に先行するため、チェックポイントの前に切捨てを行うことは不可能である。このことが、切捨てを妨げている。ダーティブロックテーブル９００を図９に示す。ダーティブロックテーブル９００の現在のコピーを、揮発性記憶装置に保守し、永久記憶装置のＲＬＯＧチェックポイントの設定プロセスの一部として定期的に記憶させることが好ましい。ダーティブロックテーブルエントリ９１０、９１１及び９１２は、回復ＬＳＮフィールド９２０及びブロックＩＤフィールド９３０を含む。回復ＬＳＮフィールド９２０の値は、動作が永久記憶装置におけるブロックのバージョンに含まれない最早ＲＬＯＧレコードを識別する。そのため、ＬＳＮフィールド９２０の値は、リドゥーすることが必要な第１ＲＬＯＧである。ブロックＩＤフィールド９３０の値は、回復ＬＳＮに対応するブロックを識別する。そのため、ダーティブロックテーブル９００は、全てのダーティブロックと、そのブロックをダーティにしたＲＬＯＧレコードのＬＳＮとを対応づける。ダーティブロックテーブル９００のもう一つのエントリは、ラストＬＳＮエントリ９５０である。このエントリに関する値は、各ブロックについて、ブロックに最新の更新を記述するＲＬＯＧレコード及びＵＬＯＧレコードのＬＳＮである。ログにおける位置を決定することが必要なため、ＤＳＬよりもむしろＬＳＮが使用される。ラストＬＳＮ９５０は、ＷＡＬプロトコルを実施するためにブロックが永久記憶装置に書き込まれた場合に、それぞれ、どれだけ多くのＲＬＯＧ及びＵＬＯＧが強制を必要とするかを示す（ＲＬＯＧに関する）ＲラストＬＳＮ９５５及び（各々のＵＬＯＧについて１つの）ＵラストＬＳＮ９５８のリストを含んでいる。そのため、ＷＡＬプロトコルを実施することは、永久記憶装置に関するブロックに組込まれた全ての動作は、安定に記憶されたＲＬＯＧ及びＵＬＯＧの両方を有していることを意味する。ＲラストＩＳＮ９５５及びＵラストＬＳＮ９５８は、（後に述べる）チェックポイントには含まれないが、それらの役割がただＲＬＯＧ及びＵＬＯＧに関してＷＡＬプロトコル実施するに過ぎないからである。そのため、好ましい実現においては、これらのエントリを回復ＬＳＮから分離しておき、それらがチェックポイント情報とともに記憶されるのを避けることが好ましい。ノードのキャッシュにおける全てのブロックに関する第１ＬＳＮが、局所ＲＬＯＧのリドゥー走査の安全点である。リドゥー回復は、安全点から順方向に局所ＲＬＯＧを読み取り、続くレコードにおける動作を再実行することにより開始される。リドゥーが必要な全てのブロックは、この走査の間に出会うリドゥーすることが必要な全ての動作を有している。先に説明したように、１－ログ想定（ａｓｓｕｍｐｔｉｏｎ）は、各ＲＬＯＧを分離して管理することを可能にする。ノードは、それ自身のＲＬＯＧを処理するだけでよく、そのため、あるノードの動作は、他のノードのキャッシュにおけるブロックがダーティである理由にはなり得ない。従って、各ブロックに対応づけられた単純な回復ＬＳＮ｛ＲＬＯＧを指定（ｎａｍｅ）しないもの｝を保存すれば充分であり、ここにおいて回復ＬＳＮは、局所ＲＬＯＧにおけるレコードを識別することが判る。

16

b．チェックポイントの設定

チェックポイントの設定の目的は、安全点の決定が、上記の様に、システムの破損の後も存続できることを確かなものにすることである。チェックポイントの設定は、安全点を移動させてリドゥーの必要なログの部分を小さくさせることのできる管理ブロック用ストラテジと組合せることができる。チェックポイントの設定のための多数の様々なテクニックがある。一つを以下に述べるが、必須のテクニックであると解するべきではない。本発明を実現する好ましいテクニックは、「ファジー」ＲＬＯＧチェックポイント設定の形式である。「ファジー」と呼ばれるのは、トランザクション又はオペレーションが

17

終了したかどうかに係わりなくチェックポイントの設定を行うことができるからである。チェックポイントを設定した情報からのダーティブロックテーブル９００のバージョンの回復は、リドゥー走査を開始する位置の決定を可能にする。ダーティブロックテーブル９００のブロックだけが永久記憶装置に記憶されていない動作を有しているため、それらのブロックをリドゥーするだけでよい。先に説明したようにダーティブロックテーブル９００は、リドゥーが必要かもしれない第１に記録されたトランザクションを示す。ＲＬＯＧによるシステム破損回復とＡＬＯＧによるメディア回復は、典型的に異なる安全点を有し、そのため切捨てられる。特に、ＲＬＯＧの切捨てられた部分が、メディア回復に必要なままであることがある。そうであれば、その切捨てられた部分は、ＡＬＯＧの一部になる。ＡＬＯＧの切捨てには、ＲＬＯＧのチェックポイントを用いる。ＲＬＯＧのチェックポイントは、チェックポイントの時点でのＲＬＯＧの切捨てを可能にする安全点を決定する。これは、永久記憶装置におけるデータの全てのバージョンが、この安全点よりも新しく、そうでなければこの点は安全ではないからである。ＡＬＯＧを切り捨てるため、永久記憶装置に関するブロックを先ずバックアップして記憶をアーカイブする。これが終了すると、アーカイブチェックポイントレコードが、了解位置（ａｇｒｅｅｄ　ｕｐｏｎ　ｌｏｃａｔｉｏｎ）、例えばアーカイブ記憶機構に書き込まれ、アーカイブチェックポイントの決定が始まった時にカレントであったＲＬＯＧのチェックポイントを識別する。ＡＬＯＧは、アーカイブチェックポイントにおいてメディア回復のために指定されたＲＬＯＧチェックポイントにより識別される安全点で切捨てることができる。永久記憶装置の全てのブロックは、ＲＬＯＧのチェックポイントが決まった（ｄｏｎｅ）後にアーカイブ記憶機構に書き込まれるため、このチェックポイントの安全点の前になされた全ての変化を反映する。ブロックのバックアップの間に、幾つかの追加のＲＬＯＧチェックポイントを採っても（ｔａｋｅ）よい。これらは、ＡＬＯＧの切捨てに影響しないか、これは、関連するログレコードか、全てアーカイブ記憶機構のブロックの状態に具体化されている（ｉｎｃｏｒｐｏｒａｔｅｄ）という保証はないからである。リドゥーをする必要はないが、ＡＬＯＧに関して残されている動作は、適用外として検知され、メディア回復処理の間無視される。チェックポイントは、ＲＬＯＧに書き込まれる。ＲＬＯＧに書き込まれた最新のチェックポイントを見出すため、その位置か、ノードに関する大域情報のエリアにおいて対応するノードの永久記憶装置に書き込まれる。最新のチェックポイント情報は、回復の間にアクセスされた最初の情報であるのが典型的である。そうでなければ、最新のチェックポイントに関するＲＬＯＧの末尾をサーチすることのできる情報である。チェックポイントは、システムがリドゥーログ、したがってリドゥー回復に必要な時間の明示制御を有するという主な利点を提供する。ＲＬＯＧが、ＵＬＯＧと組合されたとすれば、先に説明した理由で安全点をログの切捨てに使用することができない。加えて、アンドゥー情報をＲＬＯＧから排除することは、ブロックを永久記憶装置に書き込むことにより、システムがログの切捨てを制御することを可能にする。ＲＬＯＧの切捨ては、良いトランザクションの打切りを必要とするものでは決してない。このことは、取消情報を含むログを切捨てる場合には正しくはない。システムは、ブロックを後方の永久記憶装置におけるブロックの位置に書き込むことによりＲＬＯＧの制御を行う。実際、このブロックの書込は、時にはチェックポイントの部分であると考えられる。ブロックは、より古い、即ち、ＲＬＯＧにおいて更に後方の回復ＩＳＮを有している永久記憶装置に書き込んでもよい。これは、ＲＬＯＧに関する安全点をログの末尾のより近くに移動させる。オペレーションが新たに書き込まれたブロックに含まれるログレコードは、まはやリドゥー回復に必要でなく、したがって切捨てられる。メディア回復は、システム破損回復と同様の基本パラダイムに従う。ブロックのバージョンは、アーカイブ記憶機構に安定に記録される。先に説明したように、各ＡＬＯＧは、ＲＬＯＧのうちの一つの切捨てられた部分から形成される。ＡＬＯＧそれ自身は、ブロックのどのバージョンがアーカイブ記憶機構にあるかに基づき、定期的に切捨てることができる。ＤＳＩのみがブロックに記憶されており、ＩＳＮは記憶されていない場合は、どのログがアーカイブ記憶機構のブロックを更新するのに応じることのできる最新のログかを知ることもできないし、このレコードがＲＬＯＧのどこにあるかを知ることもできない。そのため、ブロックにおける情報は、ＡＬＯＧ又はＲＬＯＧを切すれるための適当な点を決定するのには不十分である。しかしながら、ダーティブロックテーブルを、ＲＬＯＧの切捨てにおけるガイドとして使用することができる。そしてＲＬＯＧの安全点を、ＡＬＯＧの安全点の設定に使用することができる。

Ｆ．ＵＬＯＧオペレーション

１．ＵＬＯＧ管理

ＵＬＯＧからのＲＬＯＧの分離が、ＲＬＯＧオペレーションを計画している（ｈａｖｅ　ｏｎ）という利点に加え、斯かる分離が、ＵＬＯＧオペレーションを計画しているという利点もある。例えば、ひとたびトランザクションがコミットしたらトランザクション特定ＵＬＯＧを破棄することができる。したがって、ＵＬＯＧに関するスペース管理は単純であり、アンドゥー情報は永久記憶装置には長く残らない。加えて、後に説明するように、アンドゥーレコードがログに永続的に書込まれることを、しばしば防ぐことができる。コミットされていないデータを含むブロックを永久記憶装置に書込む場合には、アンドゥーレコードを書込むだけでよい。リドゥー

19

ログに関しULOGとRLOGが離れていることの一つの不利益は、WALプロトコルを満たすため、コミットされていないデータを含むブロックを永久記憶装置に書込む場合に、二つのログを強制しなければならないことである。しかしながら、一般に、コミットされていないデータを有するブロックの永久記憶装置への書込は、充分に稀なので、ログの分離は、性能においてさえも、正味の利得をもたらすものである。N－ログアンドゥーに関しては、複数のノードが、ブロックにコミットされていないデータを同時に有することができる。システム破損は、これらのトランザクションが全てアンドゥーされることを必要とし、このことは、例えば、アンドゥー回復の間ロックをしてブロックのアクセスを調整することを必要とするかもしれない。全てのブロックが、アンドゥー回復に関して1－ログであることを確実にするように、あるノードからのコミットされていないデータを含むブロックは、決して第二のノードによって更新されることはない。これは、ブロックよりも小さくないロックの細分性により達成することができる。しかる後、要求ノードが、他のノードによるアンドゥー処理を決して必要としないブロックを受取る。したがって、例えば、他のノードからのトランザクションがブロックを既に更新した後に打切る場合には、そのトランザクションの影響は、既に取消されている。1－ログ取消は、複雑さを緩和するが、回復時におけるN－ログアンドゥーのシステム性能へのインパクトは、N－ログリドゥーに関するよりもはるかに少ない。これは、システム破損の時にコミットされていないトランザクションの小さなセットしか取消を必要としないからである。ブロックよりも小さくないロック細分性を有することが、同時実行を実質的に減少させることを可能にしている。本発明のテクニックは、通常は短いトランザクションに関してULOGへの書込の必要性を回避するものである。これはいかなる特定の短いトランザクションからのコミットしていないデータを有するブロックを含むキャッシュスロットも必要となることは稀だからである。斯かる稀少性の理由は、殆どの短いトランザクションは、それらのキャッシュスロットが必要になる前に、コミットするか又は打切りになるからである。取り除く（ｓｔｏｌｅｎ）べきキャッシュスロットが、コミットしていないデータを有するブロックを含んでいるとすると、WALプロトコルは、取消レコードの全ての適当なULOGへの書込を要求する。WALプロトコルは、ブロックに関するダーティブロックテーブルエントリーにおけるUラストLSNにより識別されるレコードを通して各ULOGを強制書込みすることにより、ULOGに関して強制される。先に説明したように、UラストLSNは、各ULOGにおけるブロックへの最新の更新に関しアンドゥーレコードを識別する。WALプロトコルに関して、トランザクションの更新をしていないブロックの状態を記憶させるのに必

20

要な情報は、ブロックの永久記憶バージョンを新しい状態で重ねがきする前に、トランザクションのULOGに常に永続的に記憶される。そのため、トランザクションの更新をしていないブロックの状態は、トランザクションのコミットに先立って常に永続的である。この情報は、（1）永久記憶装置のブロックバージョン内、（11）先行するトランザクションからのRLOG情報を用いる永久記憶装置におけるバージョンからの「リドゥー可能」、或いは（111）このトランザクションについてのWALプロトコルによりULOGに対してログされるか又はリドゥー回復の間につくられたアンドゥー情報を用いる（1）又は（11）により作られたバージョンからのアンドゥー回復可能、である。前の状態のままである永久記憶装置に対するブロックに関し、対応するULOGレコードのないRLOGレコードを有することが可能である。これは、「任意の」アンドゥーロギングがある場合には、ありふれたことである。斯かるブロックに関し、対応するRLOGレコードのないULOGレコードを有することも可能である。この場合ULOGレコードは無視することができる。そのため、リドゥー回復の後にアンドゥーする必要のある全ての動作が、必ずしもULOGにおいて見出される必要はない。システムが破損した場合には、失われたアンドゥーレコードを、リドゥーレコード及びブロックの前の状態から生成する必要がある。動作が、ブロックの状態を及びログされた動作の値パラメータのみによる限りは、動作が初めに行われた時に利用可能であった全ての情報が、この時点において利用可能であるため、アンドゥーレコードの生成は、可能である。　動作は、二つの理由でULOGで終了する。ブロックは、永久記憶装置に書込まれたので、WALプロトコルが、ULOGにバッファレコードを強制するか、又はWALの実施（ｅｎｆｏｒｃｅｍｅｎｔ）のためのULOGの書込か、先行するULOGリドゥー及び、ある場合には、取消バッファにある続くULOGレコードの書込に帰着するかによるものである。これらの動作に関しては、必ずしも回復の間に取消レコードを生成する必要がないのは、これらのレコードが、ULOGにあることが保証されているからである。このことが重要なのは、永久記憶装置におけるブロックのバージョンが、動作の後に来る状態を有するので、再実行をログしたトランザクション用のULOGを構成するのが不可能といえるからである。幸運なことに、これらのブロックと正確に同じであり、ULOGレコードがそのために既に存在するブロックがある。リドゥーの間、失われた取消レコードが生成される。リドゥーの終りまでに、生成されたアンドゥーレコードとULOGのアンドゥーレコードとの合併（ｕｎｉｏｎ）は、全てのコミットしていないトランザクションをロールバックすることができる。

2．　ULOGの最適化

21

本発明に関し、アンドゥーログバッファの内容を、必要な時にのみＵＬＯＧに書込むことを確実にすることにより、ＵＬＯＧの使用を最適化することができる。一般に、現在（ｃｕｒｒｅｎｔ）トランザクションからのコミットしていないデータを含むブロックを永久記憶装置に書込む場合に、アンドゥーバッファはＵＬＯＧに記憶させるだけでよい。トランザクションが、コミットされている場合には、トランザクションにおける更新を取消す必要はなく、そのためアンドゥーバッファを破棄することができる。図１０は、ＷＡＬプロトコルを用いたこのＵＬＯＧの最適化を実現するための手順のフローダイヤグラム１０００を示す。ブロックのバージョンは、永久記憶装置に書込まれるものと仮定する。書込むブロックが、コミットしていないデータを含んでいる場合には（工程１０１０）、そこでアンドゥーバッファを永久記憶装置のＲＬＯＧに書込む必要があり、いかなるアンドゥーバッファも永久記憶装置のＵＬＯＧに書込む（工程１０２０）。リドゥーバッファのＲＬＯＧへの書込及びアンドゥーバッファのＵＬＯＧへの書込（工程１０２０）の後、又はブロックがコミットしていないデータを含まない場合には（工程１０１０）。ブロックを永久記憶装置に書込む（工程１０３０）。これはＷＡＬプロトコルに従うものである。そのため、記憶すべきコミットしていないデータがある場合には、アンドゥーバッファを書込むだけである。トランザクションがコミットする各々の時、対応するアンドゥーログバッファを永久記憶装置にかきこむ必要がないので、このアンドゥーログバッファを破棄することができる。更に、取消がここでは必要ではないので、トランザクション用ＵＬＯＧ自体を破棄することができる。永久記憶装置におけるＲＬＯＧのトランザクションに関する全ての再実行レコードを記録することにより、コミットしたトランザクションか、永続的になる。その後のある時点で、更新したブロックを永久記憶装置に書込むことができる。たとえ更新したブロックを書込む前に破損があっても、ＲＬＯＧを検索してブロックの状態を再記録することができる。システムは、ＲＬＯＧのコミットレコードを記憶することにより、トランザクションがコミットすることを知覚しているのである。

３．　トランザクションの打切り

トランザクションがコミットすると、トランザクションの効果の無効化はもはや要求されないので、ＵＬＯＧはそこで破棄することができる。トランザクションの打切りに関しては、状況が幾分異なる。トランザクション用ＵＬＯＧレコードを破棄できるまでには、打切りトランザクションにより変更された全てのブロックが、無効とされたそれらの変更をしているだけでなく、その結果の無効とされたブロックの状態が、ＵＬＯＧ以外のどこかに永続的に記憶されることを確実にする必要がある。取消された状態にあるブロック自体を永久記憶装置に書込まれなければならないか（「強制」打切りと呼ばれる）、又はアンドゥートランザクションをＲＬＯＧに書込み強制しなければならない（「非強制」打切りと呼ばれる）。コミッティングトランザクションと同様に、ＲＬＯＧに対するロギング動作は、ブロックをこの場合には永久記憶装置に強制する必要を回避する。

22

ａ．非強制打切り

非強制打切りか、アンドゥーオペレーションを、先の更新の効果を反対にする打切りトランザクションの付加的動作として取扱うことにより実現する。斯かる「補償」動作は、「補償ログレコード」（ＣＬＲｓ）としてＲＬＯＧにログされる。補償ログレコードは、効率的にＲＬＯＧへと移動されたアンドゥーレコードである。しかしながら、これらのレコードを他のＲＬＯＧレコードから区別するため、余分の情報が必要である。加えて、リドゥーすべき他のログされたトランザクションに対し、ＣＬＲを正しく順に並べるため、ＳＩが必要である。図１１は、幾つかの属性を備えたＣＬＲ１０００を示す。ＴＹＰＥ属性１１１０は、このログレコードを補償ログレコードとして識別する。ＴＩＤ属性１１２０は、トランザクションに関する一意の名前である。これは、このＲＬＯＧ　ＣＬＲに対応するＵＬＯＧを見出す助けをする。ＢＳＩ属性１１３０は、先に説明したように、更新前状態識別子である。このコンテキストでは、ＢＳＩ属性１１３０は、ＣＬＲが適用された（ａｐｐｌｉｅｄ）時点におけるブロックの状態を識別する。ＢＩＤ属性１１４０は、このレコードをログした動作により変更されたブロックを識別する。ＵＮＤＯ　ＤＡＴＡ属性１１５０は、無効とすべき動作の性質を説明し、無効とすべき動作に関する充分な情報を、この動作に対応する最初の動作がブロックの状態に組込まれた後に、提供する。ＵＮＤＯ　ＤＡＴＡ属性１１５０に関する値は、ＵＬＯＧ又は取消バッファに記憶されている対応する取消しレコードからきている。ＲＬＳＮ属性１１６０は、同じ動作にとって取消しであるこの動作を説明するＲＬＯＧレコードである。この属性は、ＵＬＯＧレコードのＲＬＳＮ属性からきている。ＬＳＮ１１７０は、ＲＬＯＧにおけるその位置により識別できるため、明示的に記憶する必要がなく、ＲＬＯＧに関し一意的にこのＣＬＲを識別する。ＬＳＮは、リドゥー走査及びＲＬＯＧのチェックポイントの設定を制御するのに用いられる。トランザクションのコミットがあった場合には、トランザクションの打切りの際、トランザクションの動作を説明する全ての再実行レコードがＲＬＯＧに書込まれる。打切られたトランザクションに関しては、このトランザクションはＣＬＲに取消し動作を含んでいる。コミットに関しては、ＲＬＯＧは、トランザクション用の全ての再実行レコードが、安定に記憶されることを確実にするよう強制される。打切りに関しては、打切りは厳密には必ずしも必要ではない。必要な情報は、既にＵＬＯＧに存在してい

る。しかしながら、打切りトランザクション用のＣＬＲがＲＬＯＧに永続的に書込まれるまで、ＵＬＯＧを破棄することができない。ＲＬＯＧのＣＬＲは、しかる後ＵＬＯＧレコードの代りになる。非強制アプローチの望ましい特性は、メディア回復のために、リドゥー相のみが必要なことである。更新は、更新がＡＬＯＧマージングの間に処理されるように、適用される。全ての必要なリドゥーは、ＣＬＲを適用することにより行われるので、ＡＬＯＧを処理する際に独立したアンドゥー相が必要とされることはない。第二のテーブルは、アクティブトランザクションテーブルと呼ばれるが、アンドゥーオペレーションを行うのに必要な情報を記憶する。ダーティブロックテーブル９００のように、アクティブトランザクションテーブルは、ＲＬＯＧのチェックポイント情報の一部となるため、システムが破損しても情報が保存される。アクティブトランザクションテーブルは、アンドゥーの必要なトランザクション、アンドゥー／リドゥーロギングの状態、及びアンドゥープログレスを示す。充分な情報を、アクティブトランザクションテーブルに符合化して全てのシステム破損からの回復を確かなものにしなければならず、破損には回復そのものが行われている間に起きるものも含まれる。回復性能を改善する幾つかの情報を、更に含んでいても良い。図１２は、アクティブトランザクションテーブル１２００の例を示す。テーブル１２００は、レコード１２０５、１２０６、及び１２０７を含んでいる。各々のレコードは、幾つかの属性を含んでいる。ＴＩＤ属性１２１０は、トランザクションに関する一意の名前である。この名前は、ＲＬＯＧレコードに属するトランザクション識別子と同様のものである。ＳＴＡＴＥ属性１２２０は、アクティブトランザクションが、二相コミットの一部として「準備された（ｐｒｅｐａｒｅｄ）」かどうかを示す。複数のノードが、トランザクションに関与する場合に、二相コミットが用いられる。斯かるトランザクションをコミットするためには、全てのノードが、先ずトランザクションを準備をして（相１）はじめてノードがトランザクションをコミットすることができる（相２）。準備は、一つのノードがコミットし、もう一方のノードが打切りの場合に起きる部分コミットを回避するために行う。準備したトランザクションは、ロールバックすることが必要なことがあるので、アクティブトランザクションテーブル１２００に保存することが必要である。非準備（ｎｏｎ－ｐｒｅｐａｒｅｄ）トランザクションと異なり、準備したトランザクションを自動的に打切るべきではない。ＵＬＯＧ　ｌｏｇ属性１２３０は、トランザクション特定ＵＬＯＧの位置を示す。この属性は、ＵＬＯＧを見出す他の方法がない場合には、存在してさえいればよい。例えば、ＴＩＤ１２１０は、トランザクション用ＵＬＯＧを見出す代りの方法を提供する。ＨＩＧＨ属性１２４０は、このトランザクション用ＵＬＯＧに書込まれた取消

しレコードに関する最新の動作である動作のＲＬＯＧ　ＬＳＮを示す。このＵＬＯＧレコードは、システムの破損の後、トランザクションがコミットしていないのであれば、トランザクションをロールバックできるようにしておくため、ＲＬＳＮにしたがうＲＬＯＧレコードがリドゥーの間に生成される必要があるように、ＲＬＳＮにおけるＲＬＯＧ　ＬＳＮを含んでいる。ＮＥＸＴ属性１２５０は、取消す必要のあるトランザクションにおける次の動作のＲＬＯＧ　ＬＳＮを示す。ロールバックされないトランザクションに関し、ＮＥＸＴ属性１２５０は、トランザクションにより行われた最新の動作のレコード番号である。あるシステムは、回復の間にＣＬＲを取消すが、好ましい実施態様では、ＣＬＲＳは取消されない。そのかわり、ＣＬＲは、（ＴＹＰＥ属性を介して）タグをつけられ、そのためＣＬＲを回復の間に識別することができる。ＵＬＯＧの順序性のため、アンドゥーレコードがＵＬＯＧに強制されると、全ての先行するアンドゥーレコードも永続的となることを保証される。ＲＬＯＧは、ＵＬＯＧと同じオーダーで書込まれる。そのため、ＲＬＯＧが再実行の必要がないことが判ると、例えばその効果は既に永久記憶装置のブロックのバージョンにあるので、全ての先行するＲＬＯＧレコードは、そのためＵＬＯＧにアンドゥーレコードを有する。これは、ブロックが書込まれた場合にＵＬＯＧが強制され、そのため全ての先のＵＬＯＧレコードが同時に書込まれたために起こったのである。このトランザクションのために再実行の間に取消しレコードが生成された場合には、全ての斯かる先行レコードは既にＵＬＯＧに存在しているはずであるため、それらのアンドゥーレコードは破棄することができる。最新のＲＬＯＧに関してＵＬＯＧレコードが書込まれたそのＲＬＯＧのＲＬＯＧ　ＬＳＮは、トランザクションに関するアクティブトランザクションテーブルエントリーのＨＩＧＨ属性１２４０（図１２）に記憶される。この示されたリドゥーログレコードに先行するＲＬＯＧレコードは、再実行の間に取消しレコードを生成しないが、これはＲＬＯＧレコード全てが、既にＵＬＯＧレコードを有しているからである。ＨＩＧＨにより示されるものの一つに追従するＲＬＯＧレコードは、生成されたアンドゥーレコードを有することが必要なことがある。アンドゥーレコードの生成は、各トランザクションに関して既に適用されたアンドゥーレコードの数を注意深く監視すれば、やはり回避することができる。したがって、アンドゥー「高水位」がアクティブトランザクションテーブル１２００のＮＥＸＴ属性１２５０において符合化される。ＮＥＸＴ属性１２５０は、トランザクションのために適用される次のアンドゥーレコードのレコード数を含んでいる。正常な処理の間、ＮＥＸＴ属性１２５０は、常にトランザクションの最近のトランザクションに対するレコード数である。ＮＥＸＴ属性１２５０における値は、これらの動作がログ

25

されるにつれて増加する。アンドゥー回復の間、各々のアンドゥー動作が、その先駆アンドゥーレコードを次のアンドゥー動作として指定しながら、適用されて、そのＣＬＲがログされた後、ＮＥＸＴ属性１２５０における値は減少する。ロールバックの間にシステムの破損が起きた場合には、ＮＥＸＴ属性１２５０により示されたレコード数よりも大きいレコード数のアンドゥーレコードは、再び適用する必要がなく、そのため再実行の間に再び生成する必要がない。最終結果は、再実行の間、レコード数がＨＩＧＨ属性１２４０についての値とＮＥＸＴ属性１２５０についての値との間に下がったＲＬＯＧレコードに関し、無効レコードが生成されるものである。ＨＩＧＨ属性１２４０の値が、ＮＥＸＴ属性１２５０の値よりも大きいか又は等しい場合にはいつでも、取消しレコードを生成させる必要が全くない。

ｂ．強制打切り

「強制」打切りに関しては、ＣＬＲは書込をしない。その代り、ブロックが取消された場合は、ブロックそれ自体が永久記憶装置に強制される。この種の打切りでは、ブロックが、取消しオペレーションが行われる順序だけでなく、アンドゥーレコードを適用した結果も含むと云う知識を、ＣＬＲを書込むことなく安定に保存することが必要なことである。目標は、数個のノードが、システムの破損の結果としての一つのブロックについてのトランザクションを取消すことができるＮ－ログ取消しを支持することである。したがって、各ノードにより行われる取消しオペレーションの進行は、安定して記録されるはずである。このことは、非強制の場合においてはＣＬＲが行うことである。ＣＬＲがない場合には、何か他のテクニックが必要である。一つの代りの方法は、必要な情報を、永久記憶装置に入るブロックに書込むことである。ＣＬＲは、アンドゥー動作の完全な説明を含んでいるか、この説明の全てが必要なわけではない。強制打切りの場合において必要なことは、アンドゥートランザクションの結果及び取消しトランザクションのうちのどれが無効化されているかを記録することである。

ｇ．正常のオペレーション

正常のオペレーションの間、トランザクションの開始、ブロックの更新、ブロックの書込、トランザクションの打切り、トランザクションの準備、及びトランザクションのコミットのオペレーションは回復の要求に対してインパクトを有する。したがって、正常のオペレーションの間、回復が可能であることを確実にするためロギングに関する工程を設けなくてはならない。図１３は、トランザクションの開始オペレーションに関する手続１３００を含んでいる。先ず、ＳＴＡＲＴ　ＴＲＡＮＳＡＣＴＩＯＮ　レコードをＲＬＯＧに書込まなくてはならない（工程１３１０）。次いで、トランザクションは、「アクティブ」状態でアクティブトランザクションテーブル１２００に入る（工程１３２０）。しかる後、トランザ

26

クション用のＵＬＯＧ及びその一致が、ＵＬＯＧｌｏｃ１２３０に記録される（工程１３３０）。最終的に、ＨＩＧＨ１２４０およびＮＥＸＴ１２５０の値をゼロにセットする（工程１３４０）。図１４は、ブロックの更新オペレーションに関する手続き１４００を示している。先ず、要求される同時制御を行い、更新用ブロックをロックする（工程１４１０）。しかる後、ブロックが既にキャッシュにあるのでなければ、ブロックは永久記憶装置からアクセスされる（工程１４２０）。指示されたトランザクションは、キャッシュにおけるブロックのバージョンの直後に行われる（工程１４３０）。次いで、ブロックのＤＳＩが、動作に関するＡＳＩで更新される（工程１４４０）。その後、ＲＬＯＧレコード及びＵＬＯＧレコードは、更新用に構成され、それらの適当なバッファに通知される（ｐｏｓｔｅｄ）（工程１４５０）。ラストＬＳＮ９５０（図9）は、適当に更新される（工程１４６０）。その後、ＮＥＸＴ１２５０の値は、個の動作のためのアンドゥーレコードのためのＵＬＯＧ　ＬＳＮにセットされる（工程１４７０）。ブロックがクリーンだった場合には（工程１４７５）、ブロックをダーティにする（工程１４８０）。しかる後、ブロックをダーティブロックテーブル９００（図9）にすえる（ｐｕｔ）が、そのために回復ＬＳＮ９２０は、ＲＬＯＧのためのＬＳＮにセットされている（工程１４８５）。図１５は、ブロックがコミットしていないデータを含んでいる場合のブロックの書込オペレーションのためのフローダイヤグラム１５００を含む。先ず、ＷＡＬプロトコルを実施する（工程１５１０）。具体的には、ブロックを永久記憶装置に書込む前に、全てのアンドゥーバッファを、ブロックに関して対応するラスト用ＳＮ９５８（図9）まで書込み、ＲＬＯＧバッファを、ラストＲＬＳＮ９５５（図9）まで書込む。ブロックに関してラストＲＬＳＮにおいて識別される各トランザクションのために、これらのトランザクション用のＨＩＧＨを、ダーティブロックテーブルからのラストＵＬＳＮ属性により識別されたアンドゥーレコードのＲＬＳＮ属性におけるＲＬＯＧ　ＬＳＮの値にセットする。各ラストＵＬＳＮは、ＴＩＤ経由及びＵＬＯＧ　ＬＳＮ経由のトランザクションの両方を識別しなければならない。これらのログに関し、これらのレコードが既に書込まれているので、書込みをする必要のない時がある。しかる後、ブロックはダーティブロックテーブル９００から除かれ（工程１５２０）、ブロックは、永久記憶装置に書込まれる（工程１５３０）。ブロック書込みレコードは、その後ＲＬＯＧに書込んでブロックが永久記憶装置に書込まれていることを表示してもよいが、これは任意である。このブロック書込みレコードは、強制する必要がない。図１６は、トランザクションの打切りオペレーションに関するフローダイヤグラム１８００を含む。先ず、ＮＥＸＴ欄１２５０における値によりしめされるアンド

ゥーレコードを設置する（ｌｏｃａｔｅｄ）（工程１６１０）。その後、要求される同時制御をあたかも正常な更新により処理されているかのように正確に係わり合う（ｉｎｖｏｌｖｅｄ）ブロックについて行った（工程１６２０）。次いで、現在取消しログレコードを、その指定されたブロックに適用し（工程１６３０）、アンドゥー動作に関するＣＬＲをＲＬＯＧに書込む（工程１６４０）。その後、ＮＥＸＴ欄１２５０の値を減少させて、次の適用すべきアンドゥーログレコードを「現在の」アンドゥーレコードとして指標付けする（工程１６４０）。幾らかのアンドゥーレコードがトランザクション用のままで残っていると（工程１６６０）、制御は、工程１６１０に戻る。そうでなければ、ＡＢＯＲＴレコードをＲＬＯＧに設置する（工程１６７０）。その後、ＲＬＯＧを、永久記憶装置にＡＢＯＲＴレコードを介して記憶させる（工程１６８０）。しかる後、ＵＬＯＧを破棄する（工程１６９０）。最終的に、トランザクションをアクティブトランザクションテーブル１２００（図１２）から取り除く（工程１６９５）。図１７は、トランザクションの準備オペレーションに関するフローダイヤグラム１７００を示す。先ず、トランザクション用の準備ログレコードをＲＬＯＧに書込む（工程１７１０）。次いで、ＲＬＯＧをこの準備ログレコードに強制する（工程１７２０）。最終的に、「準備された」トランザクションの状態は、アクティブトランザクションテーブル１２００において変更される（工程１７３０）。図１８は、トランザクションのコミットオペレーションに関するフローダイヤグラム１８００を示す。先ず、トランザクション用のコミットログレコードが、ＲＬＯＧに書込まれる（工程１８１０）。次いで、ＲＬＯＧをこのレコードに強制する（工程１８２０）。しかる後、ＵＬＯＧを破棄する（工程１８３０）。最終的に、トランザクションをアクティブトランザクションテーブル１２００から取り除く（工程１８８０）。

H. システム事故回復処理

前述の説明では、ログ、状態識別子及び回復の種々の態様を説明してきた。これらは種々の方法で組み合わされて有効な回復計画になる。好ましい方法を以下に説明する。

1．分析フェーズ

分析フェーズは厳密には必要ではない。しかし、分析フェーズなしでは、或る不必要な仕事が他の回復フェーズ中行われるかもしれない。分析フェーズの目的は、最後のチェックポイントに格納されたシステム状態をシステムが事故を起こしたときのデータベースの状態にまでもたらすことである。これを行うために、ＲＬＯＧ上の最後の完全なチェックポイント内の情報が読み取られ、ダーティブロックテーブル９００（図９）及びアクティブトランザクションテーブル１２００（図１２）に対する値を初期化するのに用いられる。次いで、この最後のチェックポイントに続くＲＬＯＧ記録が読み取られる。分析フェーズは２つのテーブル上に事実上の経過記録した動作をシュミレートする。特定の記録に関して説明すると、スタートトランザクション記録がアクティブトランザクションテーブルに関するスタートトランザクションオペレーションと厳密に同一に処理される。更新ログ記録はダーティブロックテーブル９００及びアクティブトランザクションテーブル１２００に関するブロック更新と厳密に同一に処理されるが、更新は適用されない。補償ログ記録は、ＮＥＸＴ属性１２５０の値が減らされる点を除いては、ダーティブロックテーブル９００及びアクティブトランザクションテーブル１２００に関するブロック更新と厳密に同一に処理され、更新は適用されない。ブロック書込み記録に対して、ブロックはダーティブロックテーブル９００から除去される。放棄トランザクション記録に対して、トランザクションがアクティブトランザクションテーブル１２００から削除される。準備トランザクション記録に対して、アクティブトランザクションテーブル１２００中のトランザクションの状態が「準備」に設定される。委任トランザクション記録に対して、トランザクションはアクティブトランザクションテーブル１２００から削除される。アクティブトランザクションテーブル１２００中のトランザクションに対してＨＩＧＨ属性１２４０を復元するために、ＵＬＯＧはＵＬＯＧに書き込まれた最後の記録のＲＬＳＮ属性を見い出すためにアクセスされねばならない。このＬＳＮはＨＩＧＨ属性１２４０に対する値となる。他の方法としては、ＨＩＧＨ属性１２４０に対する値を使用でき又は更新できる。このＲＬＯＧ　ＬＳＮは、トランザクションのためにＵＬＯＧ上に既に記録された動作に対するアンドゥー記録を発生させるのを避けるために使用できる。この値に続くトランザクションに対するＲＬＯＧだけに対して、アンドゥー情報が発生されねばならない。したがって、ＮＥＸＴ属性１２５０は、（1）もしログ記録が更新用ならば、トランザクションに対するＲＬＯＧに書き込まれたログ記録を持つ最後の動作のＲＬＯＧ　ＬＳＮ、又は（2）トランザクションに対した書き込まれた最後のＣＬＲのＲＬＳＮ属性のいずれかである。かくして、ＮＥＸＴ属性１２５０がＲＬＯＧの分析パス中復元できる。ＮＥＸＴ属性１２５０は、ＵＬＯＧ記録内のＲＬＳＮの値を介して、実行すべき次のアンドゥー記録を識別する。ＮＥＸＴ属性１２５０は、また、無効とするためにＣＬＲが書き込まれることによって補償された動作に対するアンドゥー記録を発生させるのを避けるために用いることもできる。かくして、アクティブトランザクションテーブル１２００内のトランザクションに対するＮＥＸＴ属性１２５０より大きな、ＲＬＯＧ　ＬＳＮに対するリドゥー記録は、存在するＣＬＲが回復のリドゥーフェーズ中に加えられるときにアンドゥー操作がなされるので、リドゥー記録に対して発生されたア

29

ンドゥー情報を持つ必要がない。

2．リドゥーフェーズ

リドゥーフェーズでは、再構成されたダーティブロックテーブル９００内にダーティとして示されるすべてのブロックがキャッシュ（データ交換用メモリ）に読取られる。この読取りは、ＲＬＯＧの走査とオーバーラップしてまとめて行うことができる。幾つかのブロックが、ブロックがローカルリドゥーに関連する必要があるか否かを決定するために、数個のノードによって読み取られるが、ノードのうちの一つだけがブロックに対するリドゥーを実際に行う。しかし、このことは、ブロック書込み記録をＲＬＯＧに書き込むことによってほぼ完全に避けることができる。ブロック書込み記録は強制される必要がないので、ブロックは、このことが必要ないとき、ときには持続性記憶装置から読み取られる。しかし、このような読取りに対する不利益は小さい。各ブロックのワンログ変形版が持続性記憶装置に存在し、ただ一つだけのノードはブロックのＤＳＩに等しいＢＳＩを持つ記録をそのログに持つことができる。このノードは、ブロック上でリドゥー処理を独立に行うものである。その結果、リドゥーは、各々がそれ自体のＲＬＯＧを持つシステムの別個のノードによって並列に行われる。並行制御はここでは必要がない。リドゥーフェーズは、リドゥーを必要とするダーティブロックにアクセスし、ＲＬＯＧ記録中に示される変更を転記することによってノードのキャッシュの状態を再構成する。得られたキャッシュは事故当時の状態のダーティブロックを含む。リドゥーを受けたブロックはロックされる。得られたダーティブロックテーブル９００及びアクティブトランザクションテーブル１２００が同様に再構成される。リドゥーを受けたブロックはロックされる。分析フェーズ後のダーティブロックテーブル９００内に示されるダーティブロックに対するリドゥー記録だけが再行される必要があるかもしれない。ＲＬＯＧの反復走査はダーティブロックテーブル９００に記録された最も早期の回復ＬＳＮ９２０でスタートする。これはリドゥーに対して安全な時点である。その結果、持続性記憶装置に書き込まれてからの各ブロックに対する更新のすべてをリドゥー走査に含ませることが確保される。前述のように、ＲＬＯＧ記録をその対応するブロックに加えようとするとき、生じることがあるのは２つの場合だけである。もしＲＬＯＧ記録のＲＳＩがブロックのＤＳＩに等しくなければ、経過記録した動作が無視される。もしその代わりＲＬＯＧ記録のＢＳＩがブロックのＤＳＩに等しければ、適当な反復動作が行われる。リドゥーフェーズは履歴を反復することに関連する。ブロックの回復ＬＳＮによって示されるＲＬＯＧ記録でスタートするすべての更新ＲＬＯＧ記録は、トランザクションに属するものがその後無効とされる必要があるものであるときでさえ、加えられる。ここでの原則は、反復すべき動作に対して、初期動作が加え

30

られた状態と厳密に同一の状態でブロックに反復動作を加える必要があることである。ＲＬＯＧ記録をブロックに加える際、ブロックのＤＳＩが反復された動作に対するＡＳＩに更新される。ノードは、ＲＬＯＧ動作が加えられるとき、ブロックに適当なロックを要求する。リドゥーは、その他のノードがロックを要求しないので、ロックが許されるのを待つ必要はない。しかし、要求されたロックはアンドゥーのスタート前に許されねばならない。このことが並行制御をアンドゥーフェーズに対して初期化する方法である。もしＲＬＯＧに経過記録された通常の更新がリドゥーを必要とするならば、ＵＬＯＧ記録をそのために発生させる必要がある。ＨＩＧＨ及びＮＥＸＴの値の間のＬＳＩを持ったトランザクションに対するすべてのＲＬＯＧリドゥー記録がそれらのために発生されるアンドゥー情報を持つようになる。この情報は、好ましくは、これらの記録を識別するＲＬＳＮ属性を持ったＵＬＯＧ記録を含む。もし動作が反復されるのに要求されないならば、不適当に発生されたかもしれない早期のアンドゥー記録は、ＵＬＯＧがこの動作に対するＵＬＯＧ記録までＷＡＬを介して持続性記憶装置に書き込まれているので、破棄される。ＨＩＧＨ属性１２４０は、この時点でこの記録のＲＬＯＧ　ＬＳＮで更新され、ＲＬＯＧＬＳＮは、もしチェックポイントがとられるならば、万一現在の回復プロセスが失敗するとしても、次の回復中冗長性のアンドゥー記録発生を減らす。各トランザクションに対して、発生されたアンドゥー記録がトランザクションのＵＬＯＧバッファに記録される。これらのアンドゥー記録プラスそのＵＬＯＧ及びＣＬＲのアンドゥー記録により、能動トランザクションをロールバックすることが確保される。その結果、リドゥーフェーズの終了時には、すべての必要なアンドゥーログ記録が存在する。

3．アンドゥーフェーズ

アンドゥー回復はＮログである。その結果、アンドゥー回復フェーズは、それがトランザクションロールバック中必要とされるのと同時に並行制御を必要とする。多数のノードが同一のブロックに対してアンドゥー変化を必要とするかもしれない。通常のデータベース活動は、一旦アンドゥーフェーズが始まると、再開することができるが、通常の活動はトランザクション放棄を並行して進めることができる。適当なロッキングのすべてがこの代わりこのこと許容する。このことは、すべてのノードがリドゥーフェーズを完了するまで、アンドゥーフェーズをスタートしないことによって確保される。この結果、リドゥー中の任意のノードによって要求されたすべてのロックがアンドゥーの始まる前に適当なノードによって保持される。最初に、アクティブトランザクションテーブル１２００中のすべてのアクティブトランザクション（しかし準備されていないトランザクション）がロールバックされる。アンドゥー処理は、１つの例外を除い

31

て、厳密に放棄されたトランザクションをロールバックするのと厳密に同一に進行する。あるアンドゥー記録は、それらの記録が再行中に再生されるアンドゥーバッファ中、又は持続性記憶装置中のＵＬＯＧ中の両方に存在する。これらの複製のアンドゥー記録が検出されて無視される。このことは次のアンドゥー記録を得るためにルーチン内に閉じ込められ、その結果、事故当時に能動であるアンドゥートランザクションに対するコードの残りは、システムが通常動作するときのトランザクションを無効とするために必要なコードと実質的に等しい。これらのソース中の冗長のＵＬＯＧ記録は、すべてのアンドゥー記録が適用されるＲＬＯＧ記録のＬＳＮによって識別されるので、消去される。

Ｖ．結　論

別個のＲＬＯＧ及びＵＬＯＧを使用することにより、アンドゥー情報が絶対に必要なときにＵＬＯＧに記憶されることを確実にすることによりロッギングオペレーションの最適化が可能となる。必要性が生じるときに対する試験は、非委任トランザクションに関連する変化に必要な情報のすべてが記憶されるか又は再形成されるかである。さらに最適化は回復中になされる変化のカウントを保持することによって得られる。変形及び変更がこの発明の精神及び範囲から逸脱することなくなされうることは当業者にとって明らかである。例えば、図１に示すアーキテクチャは異なったものでもよく、各ノードに割り当てられるアンドゥー及びリドゥーログの数は変更できる。本発明はそのような変形及び変更を含むものであり、そのような変形及び変更は添付した特許請求の範囲の範囲内に入るものである。

【図面の簡単な説明】

【図１】本発明の実施例なためのコンピュータシステム

32

を示す概略図、

【図２】ブロック及びページを示すディスクの一部の概略図、

【図３】レドログの概略図、

【図４】アンドゥーログの概略図、

【図５】アーカイブログの概略図、

【図６】レドオペレーションの実行のためのフローチャート、

【図７】クラッシュからの回復を実行するためのフローチャート、

【図８】アーカイブログを組合せるためのフローチャート、

【図９】ダーティーブロックテーブルの概略図、

【図１０】アンドゥーログの使用を最大利用すべくライトーアヘッドプロトコルを実行するためのフローチャート、

【図１１】補正ログレコードの概略図、

【図１２】アクティブトランザクションの表、

【図１３】トランザクション開始オペレーションのためのフローチャート、

【図１４】ブロック更新オペレーションのフローチャート、

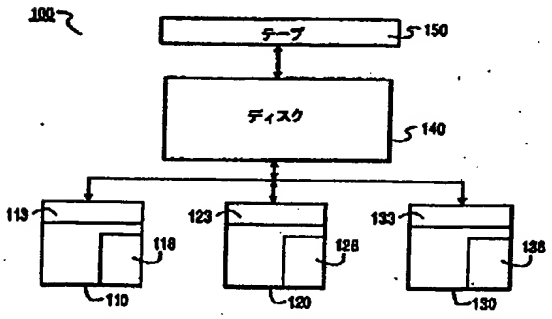
【図１５】ブロック書込オペレーションのフローチャート、

【図１６】トランザクションの放棄オペレーションのためのフローチャート、

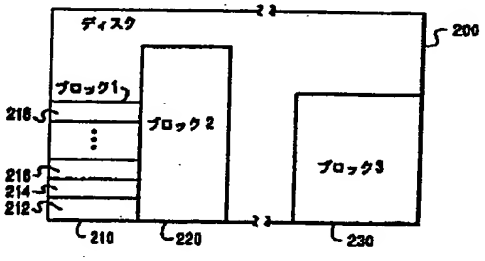
【図１７】トランザクションの準備オペレーションのためのフローチャート、

【図１８】トランザクションの変更オペレーションのためのフローチャート。

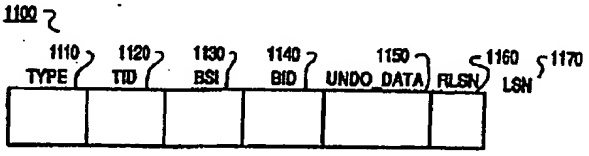
【図１】



【図２】

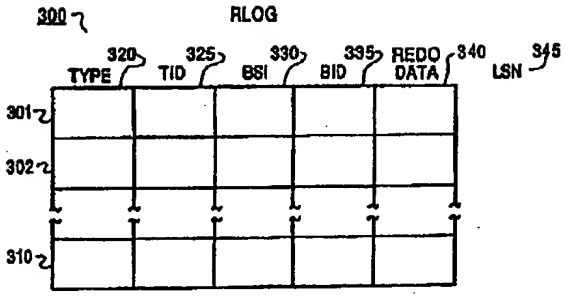


【図１１】

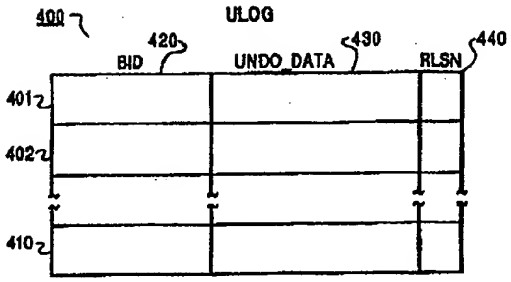


LOG記録の比較

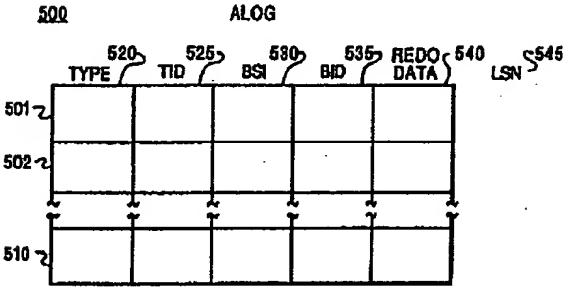
【図3】



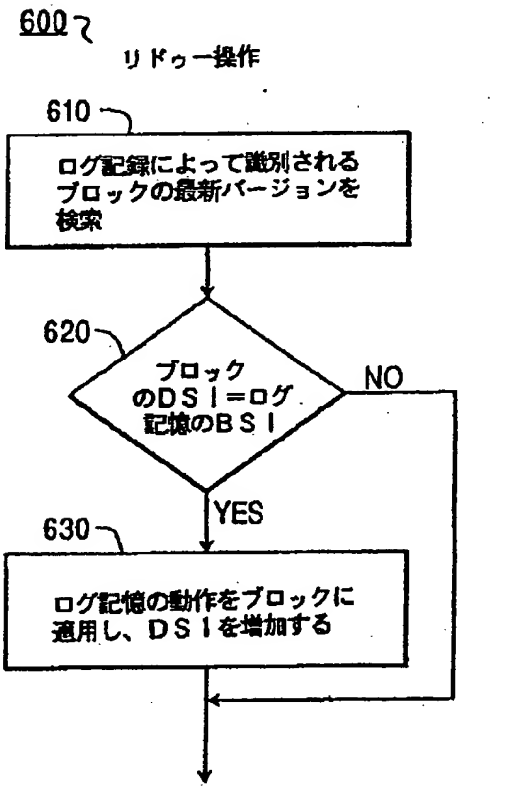
【図4】



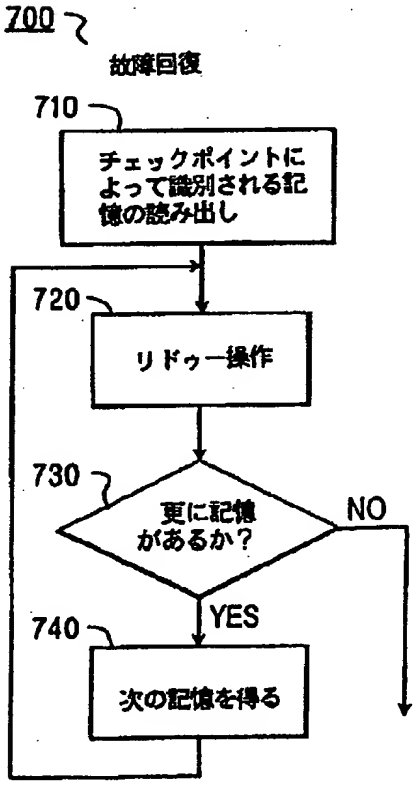
【図5】



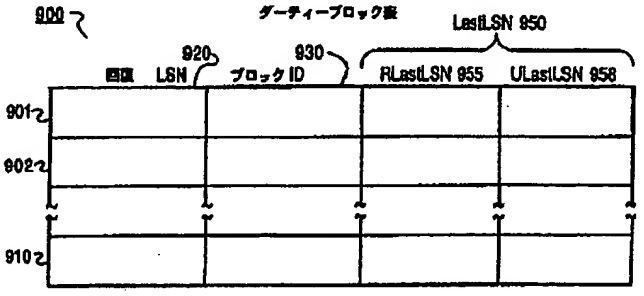
【図6】



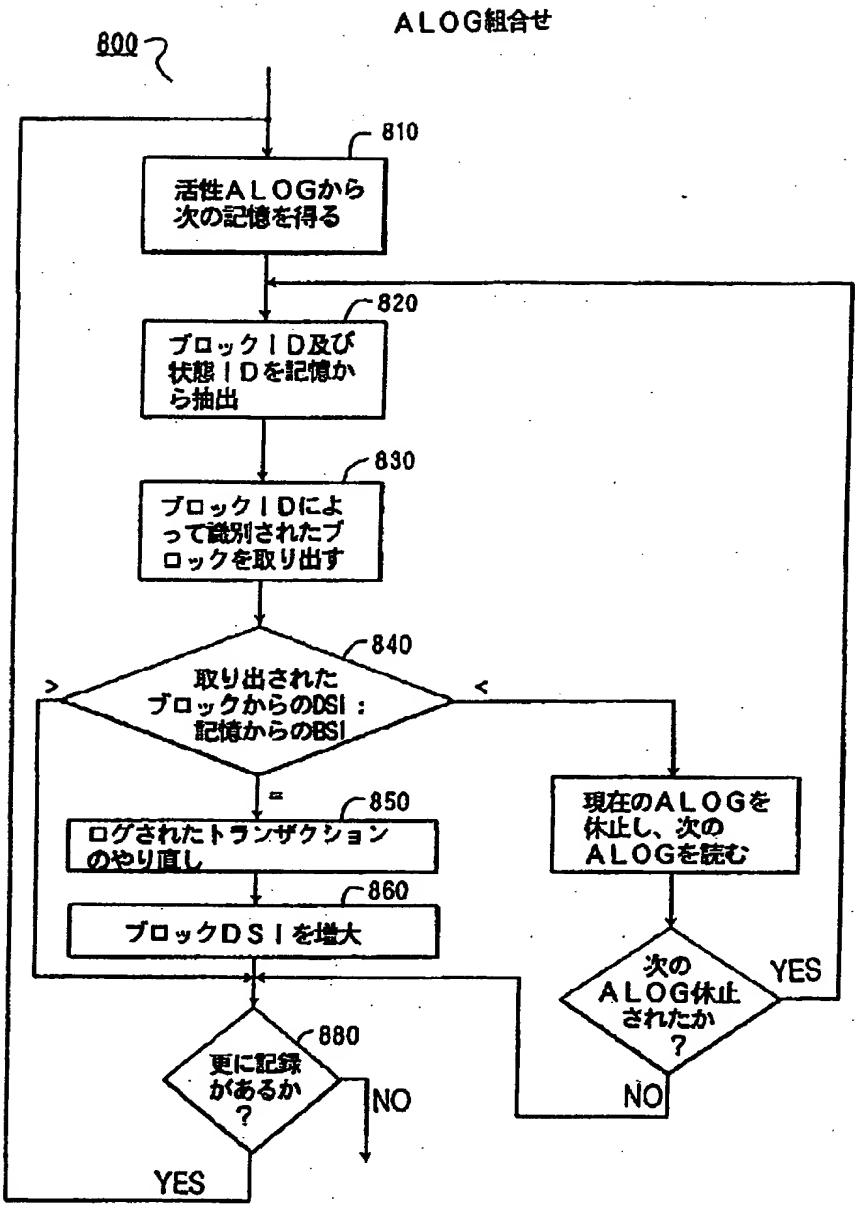
【図7】



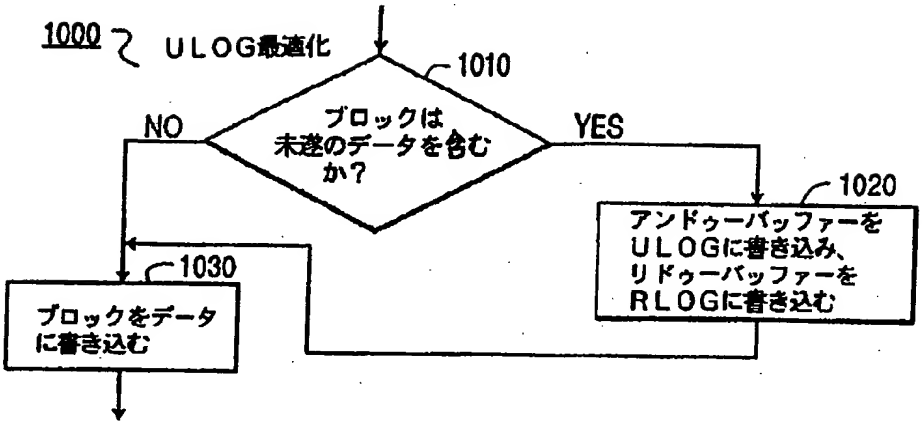
【図9】



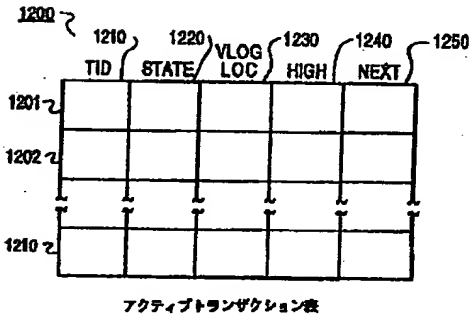
【図８】


ＡＬＯＧ組合せ
800
810
活性ＡＬＯＧから次の記憶を得る
820
ブロックＩＤ及び状態ＩＤを記憶から抽出
830
ブロックＩＤによって識別されたブロックを取り出す
840
取り出されたブロックからのDSI：記憶からのBSI
>
=
<
850
ログされたトランザクションのやり直し
860
ブロックＤＳＩを増大
880
更に記録があるか？
NO
YES
現在のＡＬＯＧを休止し、次のＡＬＯＧを読む
次のＡＬＯＧ休止されたか？
YES
NO

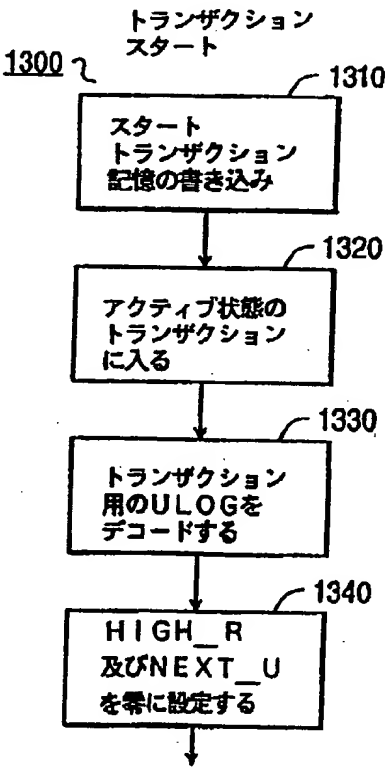
【図１０】



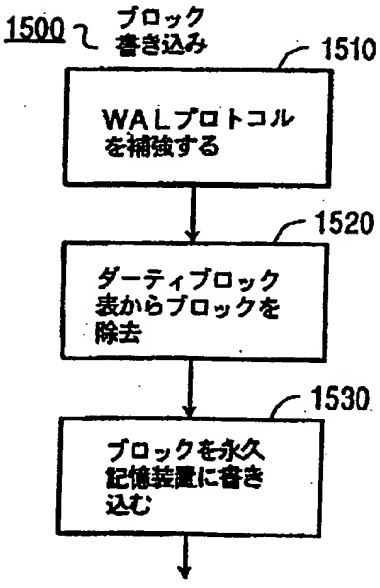
【図１２】



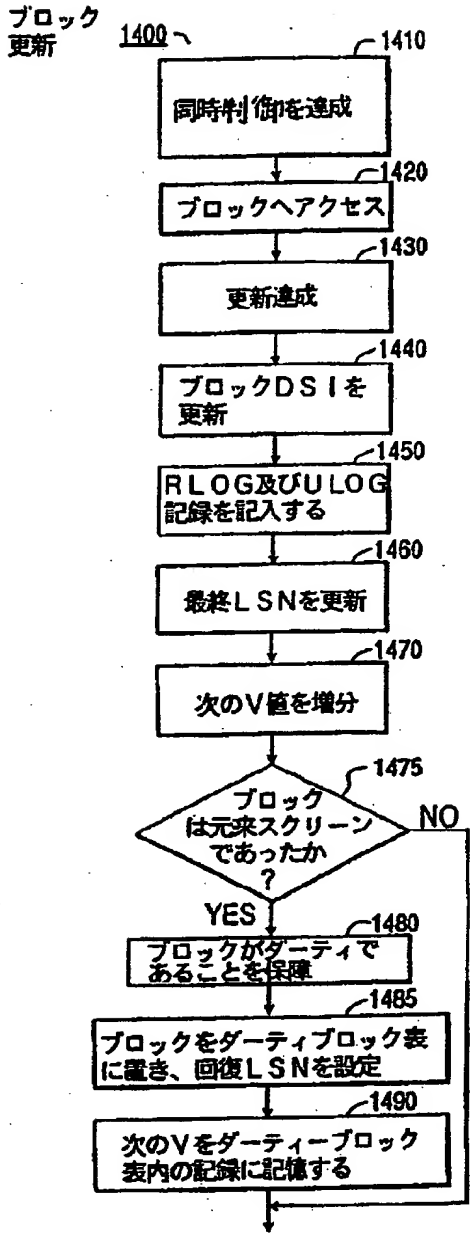
【図１３】



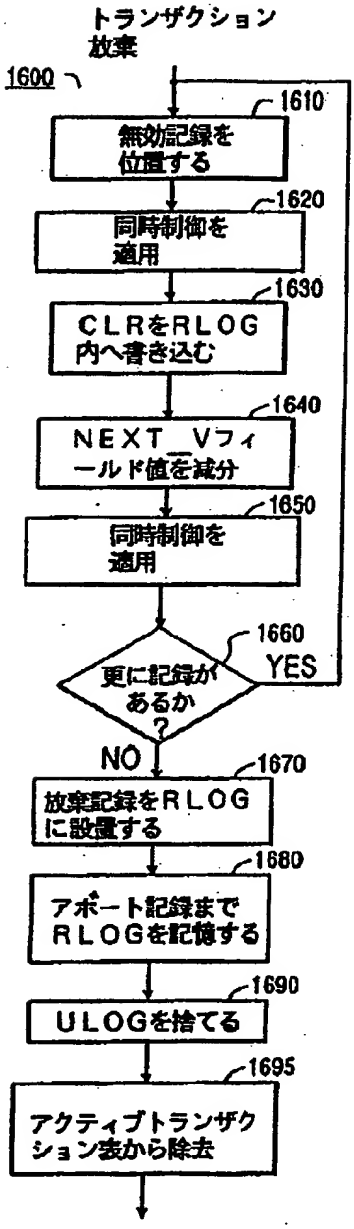
【図１５】



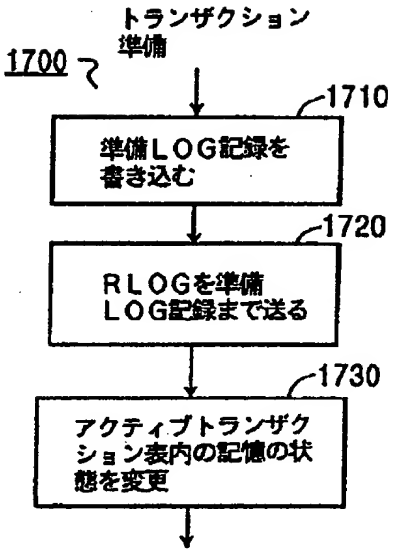
【図１４】


ブロック更新
1400
1410
同時制御を達成
1420
ブロックへアクセス
1430
更新達成
1440
ブロックDSIを更新
1450
RLOG及びULOG記録を記入する
1460
最終LSNを更新
1470
次のV値を増分
1475
ブロックは元来スクリーンであったか？
NO
YES
1480
ブロックがダーティであることを保障
1485
ブロックをダーティブロック表に置き、回復LSNを設定
1490
次のVをダーティーブロック表内の記録に記憶する

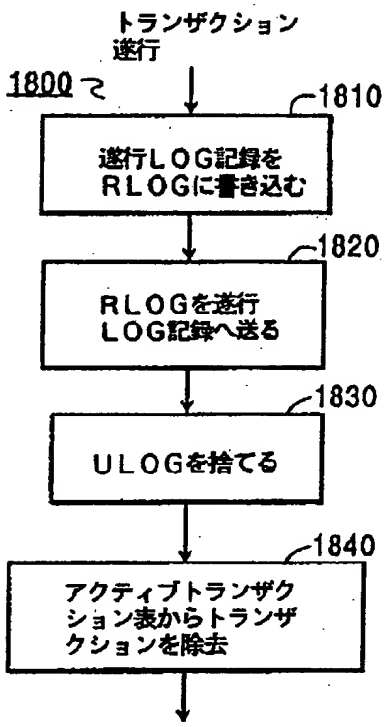
【図１６】


トランザクション放棄
1600
1610
無効記録を位置する
1620
同時制御を適用
1630
CLRをRLOG内へ書き込む
1640
NEXT_Vフィールド値を減分
1650
同時制御を適用
1660
更に記録があるか？
YES
NO
1670
放棄記録をRLOGに設置する
1680
アボート記録までRLOGを記憶する
1690
ULOGを捨てる
1695
アクティブトランザクション表から除去

【図１７】



【図１８】



フロントページの続き

(72)発明者　ピーター　エム　スピロ
アメリカ合衆国　ニューハンプシャー州　03062　ナシュア　ウェイクフィールド　ドライヴ　1
(72)発明者　アショーク　エム　ジョッシー
アメリカ合衆国　ニューハンプシャー州　03062　ナシュア　セント　ジェームズ　プレイス　718
(72)発明者　アナント　ラグハーヴァン
アメリカ合衆国　ニューハンプシャー州　03062　ナシュア　ビー　ベイ　リッジ　ドライヴ　26
(72)発明者　ティルマンジャーナム　ケイ　レンガラジャン
アメリカ合衆国　ニューハンプシャー州　03062　ナシュア　6　ハムレット　ドライヴ　5